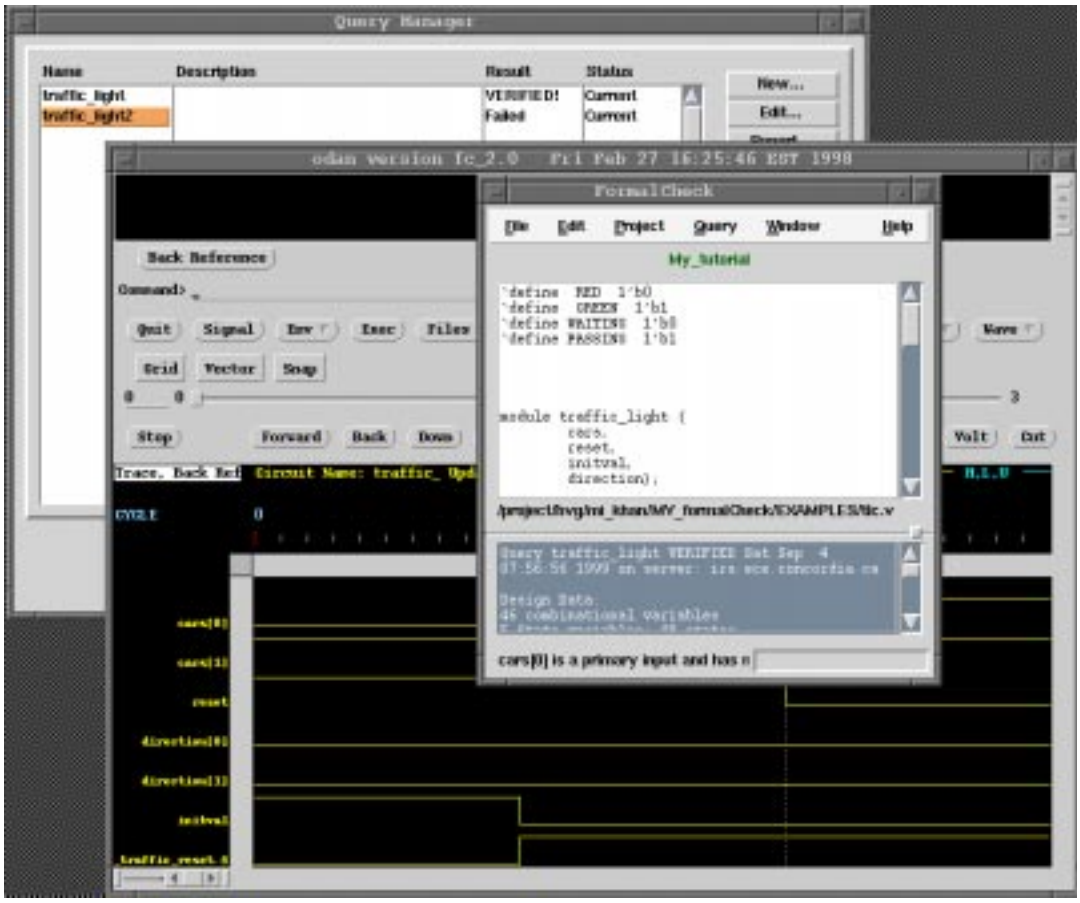
Hands-on Manual to FormalCheck Version 2.3.2

Hardware Verification Group

FormalCheck



Concordia University
Department of Electrical and Computer Engineering
Montreal, Quebec, Canada.

July 2008

TABLE OF CONTENTS

Page No.

Preface 5

Audience 5

1. **Introduction** 6

2 **Verifying Techniques used in FormalCheck** 9

FormalCheck Project 9

Queries (properties + constraints) 9

Pros and Cons of using Constraints 16

Checking for Overconstraint 16

3 **Stepwise Procedure of Verifying a Design in FormalCheck** 17

Step 1 (Initializing Step) 17

Step 2 (Saving the Project) 18

Step 3 (Opening a file: optional) 18

Step 4 (Compiling the Verilog/VHDL design - BUILD) 18

Step 5 (Adding a Constraint). 19

Step 6 (Create/Edit Query) 20

Step 7 (Verify Query) 21

4 **Post-Processing Analysis** 22

5 **Advanced Options** 24

New Features 25

AutoCheck 25

Macro 25

Page No.

Clock Extraction: 25

Script Interface 25

IMPORTANT BUILD OPTIONS 26

ADVANCED QUERY RUN OPTIONS: 26

THREE VERIFICATION ALGORITHMS 26

A Recommended Procedure for Verification 27

References 28

APPENDICES 29

APPENDIX A 29

The System Settings to Run FormalCheck 29

APPENDIX B 30

FormalCheck File Description 30

Contents of the file README 30

Contents of the file verify.stdout. 31

Contents of the file verify.out 33

Contents of the file query.c 34

APPENDIX C 35

Design Tips (for verification purpose) 36

APPENDIX D 36

The “arbiter” example 36

Description of the Arbiter to be verified 36

Arbiter Basic Specification 38

RTL design 38

Page No.

VERIFICATION RESULTS 48

Query-1 48

Query-2 49

Query-3 51

Query-4 52

Query-5 55

.

Preface

Audience:

This guide is written for research students and design engineers who will use FormalCheck as a tool of hardware verification for the first time. It summarizes the available material (FormalCheck User Guide, FormalCheck on-line Manual, etc.) that the manufacturer has provided with the tool.

1. Introduction

Formal verification means a mathematical proof which assures that a property holds of a design model. The need for "correct" designs in *safety-critical applications*, coupled with the major *cost associated with products delivered late*, are the two of the main reasons behind the recent popularity of **formal hardware verification**. The verification methods require considerable time and expertise to verify even fairly simple systems. As a result, practical application has been limited to a few domains, such as *security* and *safety-critical* systems, where ethical or legal requirements demand the highest assurance of correctness, regardless of the cost [1].

Two well-established approaches to verification are, model checking and theorem proving. Theorem proving means that, given a set of axioms and a theorem formulated in some logic, there exists a proof generated by the inference system. On the other hand, model checking is a technique that relies on building a finite model of a system and checking that a desired user-defined property holds in the model [2]. FormalCheck is a model checker, designed to help alleviate the functional verification bottleneck.

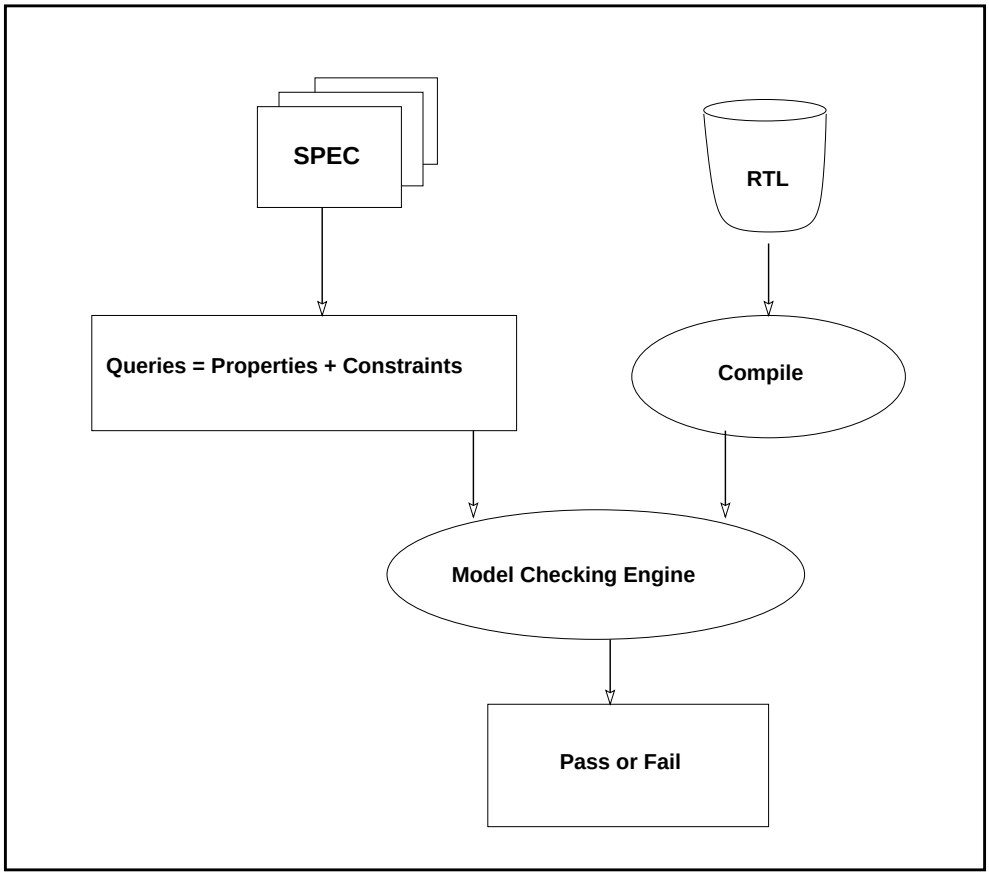


FIG 1.1: FormalCheck Verification procedure [3]

FormalCheck supports the synthesizable subsets of Verilog and VHDL hardware design languages. The source code does not need to be modified for the sake of verification. The user sup-

plies FormalCheck with a set of queries to be verified on the design model. The queries are simply statements (formalizations) of important behaviors described in the specification.

FormalCheck aids the user by automatically back referencing each error to the offending line to the source code. The errors can be fixed and the source recompiled without leaving the tool. The tool explores all possible input scenarios, guaranteeing that no bugs are missed for lack of vectors.

It is common to wait for the test bench to be completed before starting simulation and a complete test bench is only created at the chip level. With FormalCheck, verification can be started as soon as the first block is designed and the first query is written. This speeds up the design cycle by catching the bugs at an earlier stage and also reduce the verification effort. In short, “Formal-Check model checker can be used when the design is fluid or only partially defined” [3].

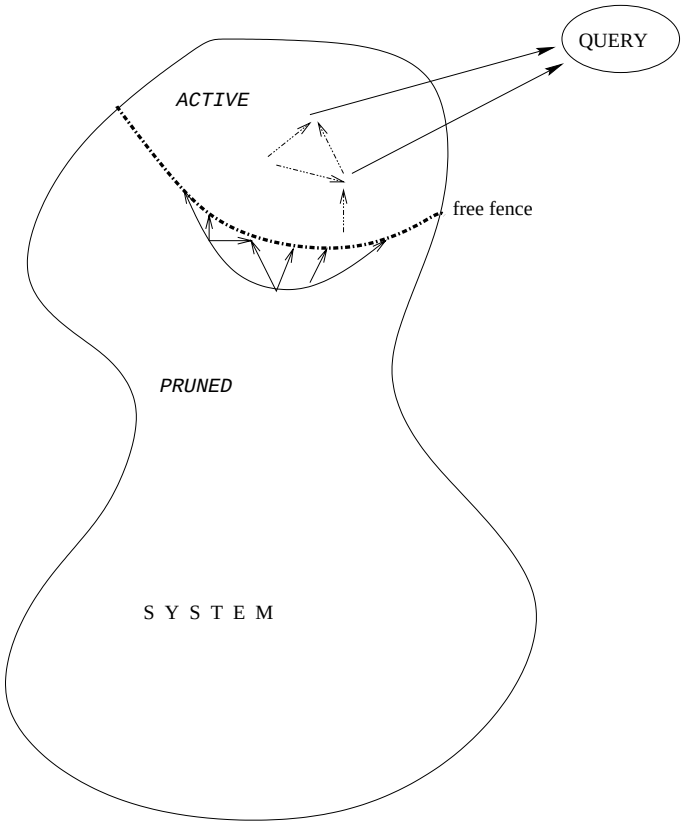


FIG 1.2: *Localization Reduction algorithm*[3]

“It is of paramount importance that the tool be able to reduce the model automatically relative to the property under check, to the greatest extent possible” [3]. FormalCheck uses two approaches of reduction algorithm (*1-step* and *iterative*). Most of the reductions tend to be property-dependent *localization reductions* [5], in which the part of the design model that are irrelevant to the property being checked, are abstracted away. In FormalCheck, localization reduction is applied dynamically. At each step of the algorithm, the model is adjusted by advancing its “free fence”

(please refer to the Figure 1.2) of induced primary inputs, in order to discard spurious counter examples to the stated query [5].

1-step reduction looks at the dependency graph and removes any portion of the design from the verification proof that cannot affect the outcome of the query. This algorithm propagates constants contained in the design or in the input assumptions to further reduce the design.

An additional level of reduction can be achieved by *Iterative Reduction* algorithm. It takes an attempt to find a small portion of the design that can be used to verify the current query. This technique guarantees that queries proven to be true on the small portion of the design would also be true for the entire design. This algorithm can be run with or without a user-supplied starting point (reduction seed) and can reduce the *state space* of the design by several orders of magnitude, allowing *larger designs* to be verified.

This tool checks if a change (by the user) could affect the outcome of the verification. If not, it marks the query as proven by the previous runs. This unique capability reduces the *regression time*. Through FormalCheck's Back-Reference utility, a click on an error in the waveform pops up the source with the cursor on the line that caused the error.

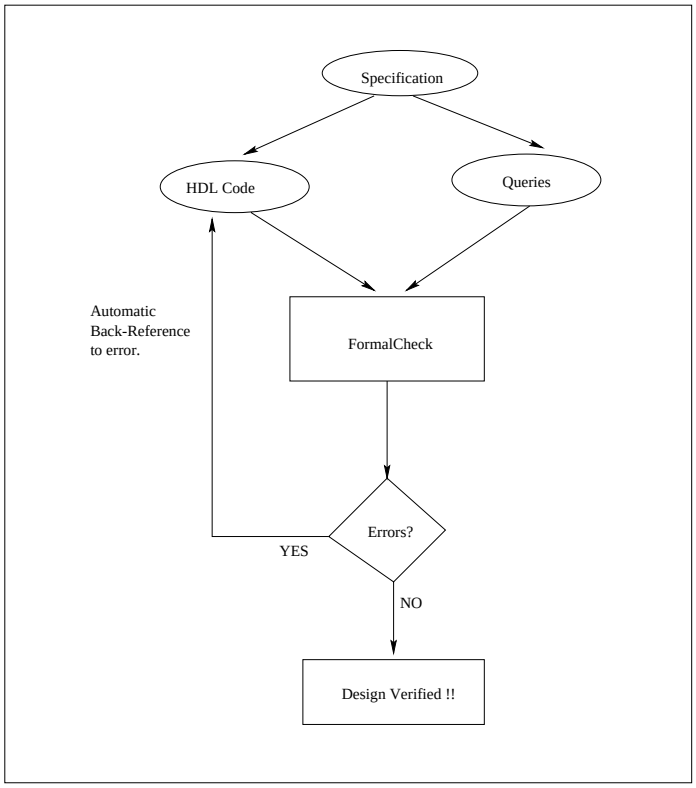


FIG 1.3: FormalCheck Verification Cycle [6]

2. Verifying Techniques used in FormalCheck

To use either the Graphical User Interface (GUI) or the Command Line Version, one must aware of the functionality of the each step to wisely use the different advanced options provided by FormalCheck. The following literature refers to the GUI version of FormalCheck tool.

Note: If anyone already has some experience with FormalCheck, can skip this chapter and can have a look at the chapter 4 (stepwise procedure) in order to just remind him/herself of the tool

FormalCheck Project:

A FormalCheck *project* is a container (directory structure) that holds all information relevant to the verification [6]. One may start the tool either by opening an existing project or by creating a new one. A project (.fpj) file contains the following information:

- pointer to the model design.
- necessary information to compile the design (i.e. HDL language, top level entity/module name, hierarchical dependencies, etc.).
- data structure built for the verification.
- queries (properties + constraints)
- result of the verification of the queries.

Queries (properties + constraints):

Existing model-checkers use some form of CTL (Computation Tree Logic) to define properties [3]. The idea of using a logic was discarded my FormalCheck, because this approach is sometimes hard to be defined for the general users. In FormalCheck, each **property** is defined using one of a small set of templates, *each with a clear intuitive and simple semantics* and correctly as impressive as the class of **omega** (ω) **automata**. Of course, what is gained in simplicity is lost in flexibility [3].

A property is verified if and only if no failures are found after exploring all reachable states and possible input combinations [6].

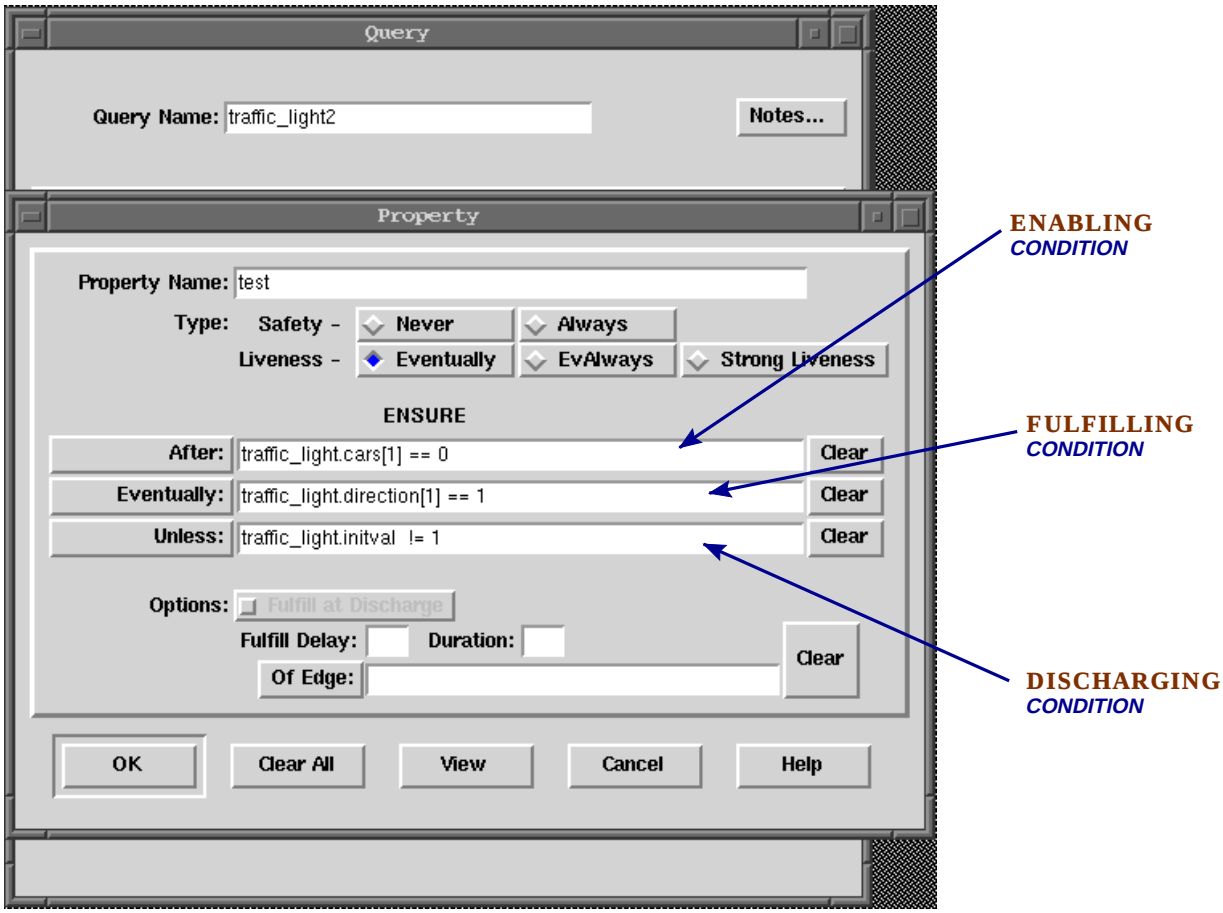


FIG 2.1: *Property Panel*, showing essential features.

The above figure defines a property which checks that after each time the designated *enabling condition* is enabled, the designated *fulfilling condition* holds continuously unless the *discharging condition* becomes true [3].
"**Fulfill at Discharge**" option is used when it is necessary to require the Fulfilling Condition concurrently with the Discharging Condition.

Table 1: Explanation of the option "Fulfill at Discharge"

Regular Fulfil and Discharge condition	With option: Fulfill at Discharge (Figure 2.1)
Example: Always (A) *Fulfilling Condition* Unless (B=1) *Discharging Condition*	Example: Always (A) *Fulfilling Condition* Unless After (B=1) *Discharging Condition*
Explanation: The Fulfilling Condition will discharge when B=1.	Explanation: The Fulfilling Condition will discharge when B becomes equal to 1, while A still holds.

Property (safety, liveness):

1. **Safety**-This kind of property ensures that nothing "bad" happens. The Safety property fails if the undesired behavior is exhibited in any state which is reachable with the phase state active [6].

   There are two kinds of Safety property in FormalCheck: a) Never b) Always

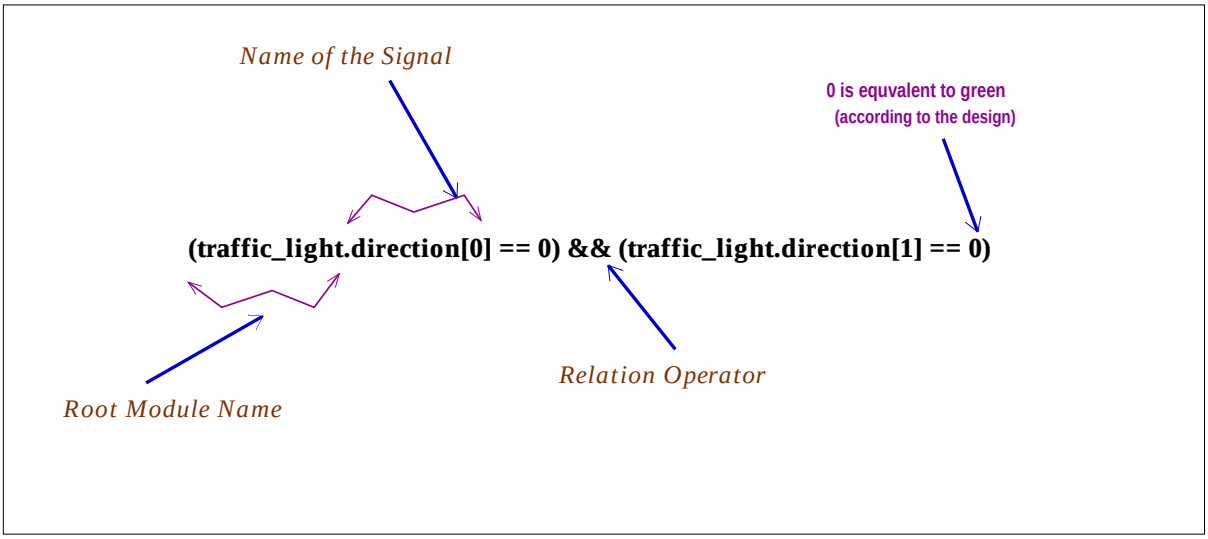

Name of the Signal
0 is equvalent to green
(according to the design)
(traffic_light.direction[0] == 0) && (traffic_light.direction[1] == 0)
Relation Operator
Root Module Name

FIG 2.2: A property in FormalCheck: "Light is never green in both directions"

a) Never: "FormalCheck will explore all possible inputs and reachable states to verify that the condition can NEVER occur" [6].

Figure 2.2 shows, how one can specify a property in FormalCheck. This can be done either my writing manually or with the help of buttons (signal selection).

b) Always: "FormalCheck will explore all possible inputs and reachable states to verify that the condition can NEVER be false" [6].

2. **Liveness**- This type of property guarantees that eventually something "good" happens. "*A Liveness property fails if there exists a sequence of inputs which can postpone the required behavior indefinitely.* Since all designs are finite state, a liveness failure amounts to finding a cycle of reachable states where the required behavior does not occur" [6]. FormalCheck defines three kinds of liveness property:

   a) Eventually
   b) Eventually Always
   c) Strong Liveness

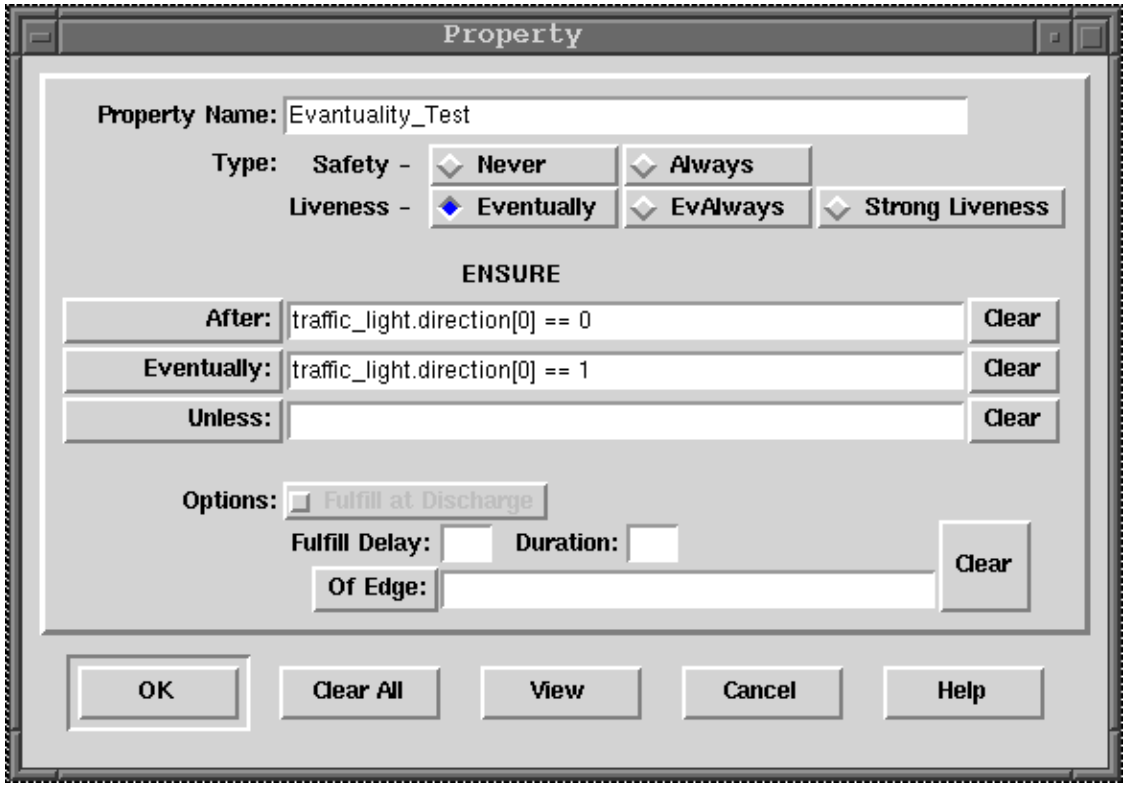


FIG 2.3: Property panel with an Eventual Property.

a) Eventually: The Fulfilling Condition must eventually become true. However, it does not have to remain true all the time. The above eventual liveness property can be written in words as follows:

After (direction(0) = GREEN)
Eventually (direction (0) = RED)

b) Eventually Always: This is like steady state "Eventually" property. The Fulfilling Condition must remain true (Steady State) at the time of discharging. "It does not have to reach steady state within any time limit and may become true and then false several times before it reaches the steady state" [6].

c) Strong Liveness: Strong liveness differs from the other liveness property by its Enabling Condition. Here, the Enabling Condition is allowed to be checked repeatedly upon the failure of fulfilling condition.

Table 2: The Difference Between Strong Liveness and Eventually

Strong Liveness	Eventually
Format: If Repeatedly (event A) *(Enabling Condition)* Eventually (event B) *(Fulfilling Condition)*	Format: After (event A) *(Enabling Condition)* Eventually (event B) *(Fulfilling Condition)*
The Enabling Condition need to be satisfied along the failure cycles.	The Enabling Condition need not be satisfied along the failure cycles. The Enabling Condition may be satisfied before the failure cycle is entered.
Both Enabling and Fulfilling Condition are required for Strong Liveness property.	When Enabling Condition is absent, the Fulfilling Condition is active until it is discharged (if ever).
Computationally not simple	Computationally simpler.

Constraint: In FormalCheck, design constraints are defined using a companion set of templates (property templates and constraint templates are paired), and each check on a design model is performed in the context of a set of properties and constraints, termed "**query**" [3]. All the constraints belonging to the project are listed in the **Constraint Library** panel (please refer to Figure 3.3). Constraints limit the state space of a design model.

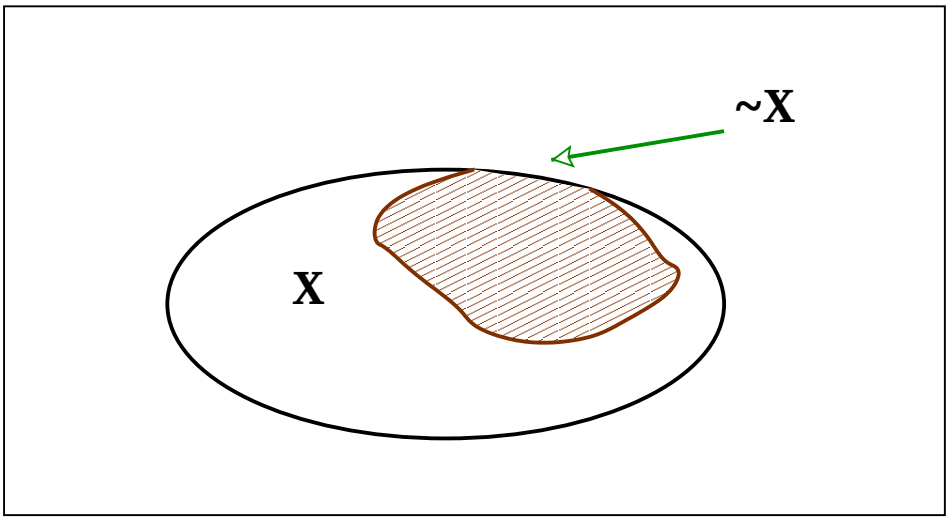


FIG 2.4: The Effect of Constraints in the Reachable State Space [6]

Let us assume that, the Figure 2.4 is the total reachable space of a variable X for some design model. The constraint to be added as "X is true always". The area with the thick boundary is then excluded from verification as it belongs to negative X. If we put constraints to some signals which includes a feature of the design, then that will not be verified and this situation is called "Overconstrained".

WARNING: We should be careful about the model being not OVERCONSTRAINED.

There are 2 types of general constraints:

i) Safety (never, always)
ii) Fairness (eventually, strong fairness)

Safety constraints are the constraints applied with the safety property and the fairness constraints are applied with the liveness property of the query.

Constraint SHORTCUTS: Constraint Shortcuts are provided by the tool to use as ready-made constraints. There is also "general" constraint in order to define any other kind of constraints. Shortcuts are only allowed for the primary inputs.

i) Assignment
ii) Clock
iii) Constant
iv) Reset/Repeat

i) Assignment: An input signal always takes the value of an expression. The expression may contain internal signals, but the signal must be a primary input.

ii) Clock: This is a very important constraint. It is an oscillating signal and FormalCheck detects the rising and falling edge of the signal. Example, *design.CLK == rising* *(Verilog)*

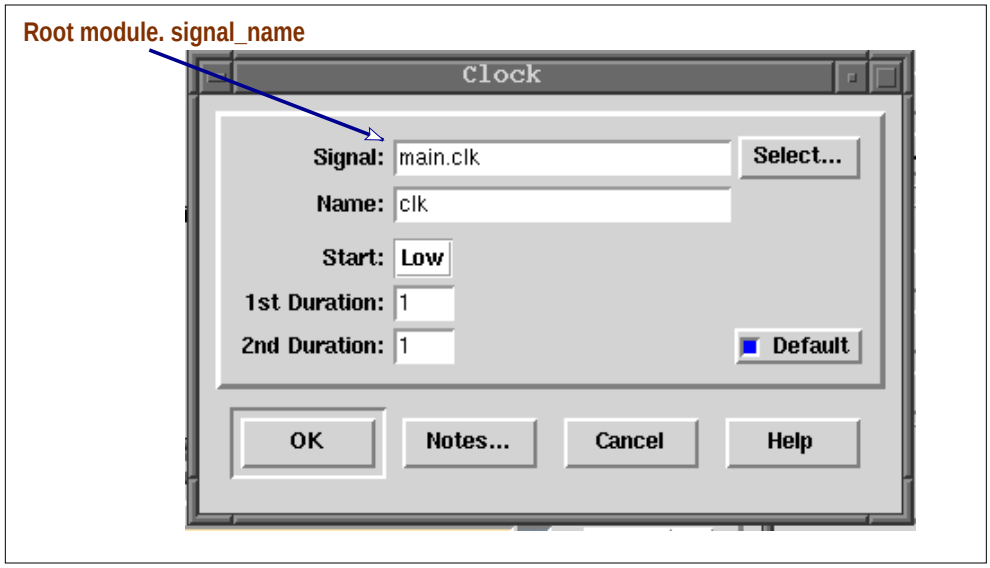


FIG 2.5: The usage of Clock Constraint

During the verification, this constraint can simulate the behavior of the clock ticks.

iii) Constant: It assumes a signal to have a constant value.

iv) Reset/Repeat: This a very useful constraint. It is used to constrain a signal to a user-defined waveform. The waveform may be an one shot waveform or a repeat (periodic) one. A signal is first selected, then a waveform is specified. Each state in which the selected signal matches, the specified waveform is excluded from the verification. This constraint can wisely be used for the initialization of all the input signals. But to do that one should always remember that it is going to be needed to add some lines code to the original design model.

Example: The following is a Verilog example (portion of a design). The reset/repeat constrain can be applied to the signal reset_elevator (please refer to the Figure 2.6) to initialize the signals "direction" and "movement".

```
 // initialization block
always @(reset_elevator)
begin
    if (reset_elevator)
    begin
        direction = 'UP;
        movement = 'STOPPED;
        end
end
```

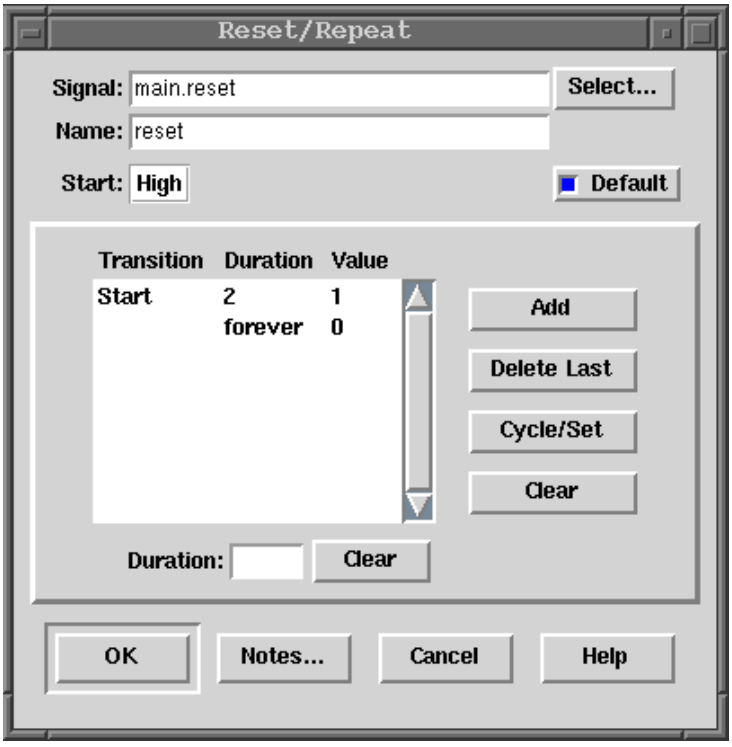


FIG 2.6: The usage of Reset/Repeat Constraint

The settings on Figure 2.6 will produce a **main_reset** signal (passed as a parameter while instantiating another module instance, and thus, is equivalent to **reset_elevator**) which is "high" (logic value 1) for 2 cranks (unit of time) and "low" (logic value 0) for the rest of the verification period. As the block is active only at positive values of the signal reset_elevator, all the input signals in the block are initialized to the desired values.

Cranks: This is the term FormalCheck uses for "unit of time".

Pros and Cons of using Constraints:

(+) Constraints limit the state space of a design to be verified.
(+) Constraints can significantly reduce the run time.
(+) One can limit the input by using constraint on it, and thus restrict the design. Any error in the restricted design is also an error for the total design. So, running a severely restricted design can give a good *early warning* concerning the computational complexity of the unrestricted design.
(-) *Mutually inconsistent* constraints can compromise the verification by preventing an important design flaw being exposed.

Mutually inconsistent: While individually taken, works fine. But simultaneous use impose contradictory results.

Checking for Overconstraint:

When the properties are overconstraint, the verification result comes as "Vacuous". It means, the Enabling Condition of the property was never satisfied. Because,

1. The Enabling Condition was wrongly formed.
2. The verification engine could not reach a state where the Enabling Condition was satisfied.
3. Inconsistent definitions while forming constraints. Like, X=1 && X=0.

The following are some procedures to solve problems with constraints:

- The only conclusive way to check that a design is not overconstraint is to convert all constraints to their corresponding properties and verify these properties on the model that derives the inputs of the given design,
- If -DVACCHECK (please refer to page 8-8 on FormalCheck User's Guide, reference no. 6) fails, one can apply -DVACCHECK **"run option"** (Query panel) to successive subsets of the constraints until a small mutually inconsistent subset is identified.
- One can check "Missing value Report" for values some signals never attain.

3. Stepwise Procedure of Verifying a Design in FormalCheck

Step 1 (Initializing Step).

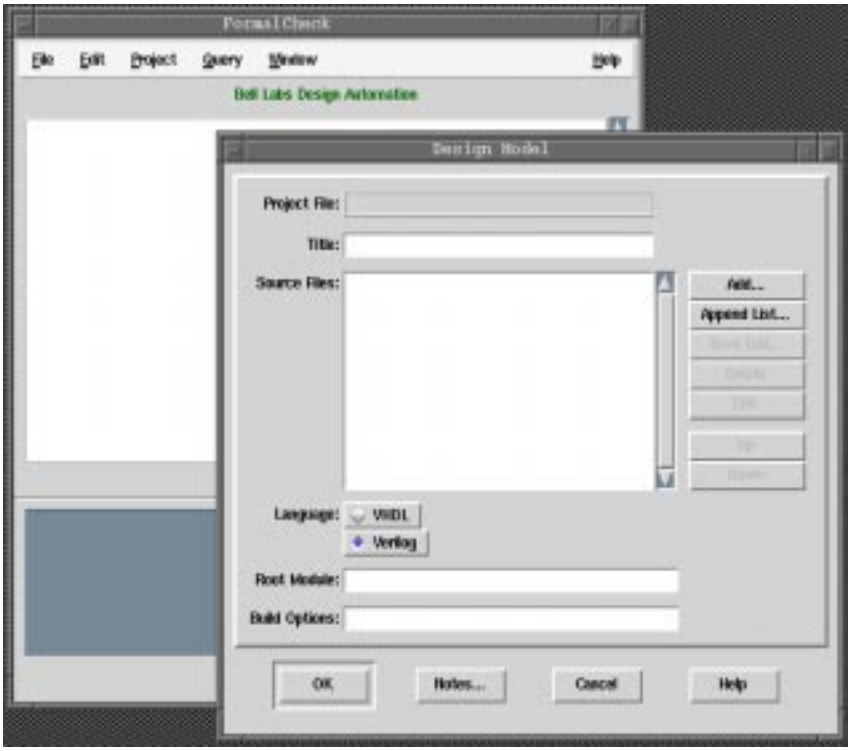


FIG 3.1: Design Model Panel (for *Step 1*)

PROJECT

NEW/OPEN

This will invoke "**Design Model Panel**".

- Name of the project has to be provided. ("Title").
- Design language has to be chosen (VHDL or Verilog).
- For VHDL (.vhd/.vhdl): Root Entity (mandatory)
  Architecture (mandatory)
- For Verilog (.v): Root Module (mandatory)
  Build Option (optional)

ADD

This will invoke "Select Design File Panel".

- A design file (same language chosen before) has to be selected.
- Option "SAVE LIST": This will create a text file that contains a list of file names of the design model files that are currently displayed on the panel.
- Option "NOTES": This will allow to write notes about the project.

Step 2 (Saving the Project).

PROJECT

SAVE / SAVE AS

FormalCheck uses the extension "**.fpj**" for the files containing the project. The user has to enter a name for the file.

Step 3 (Opening a file: optional).

FILE

OPEN

This is not a mandatory step for the verification process. But, it allows to debug syntax error (edit) the design (Verilog/VHDL).

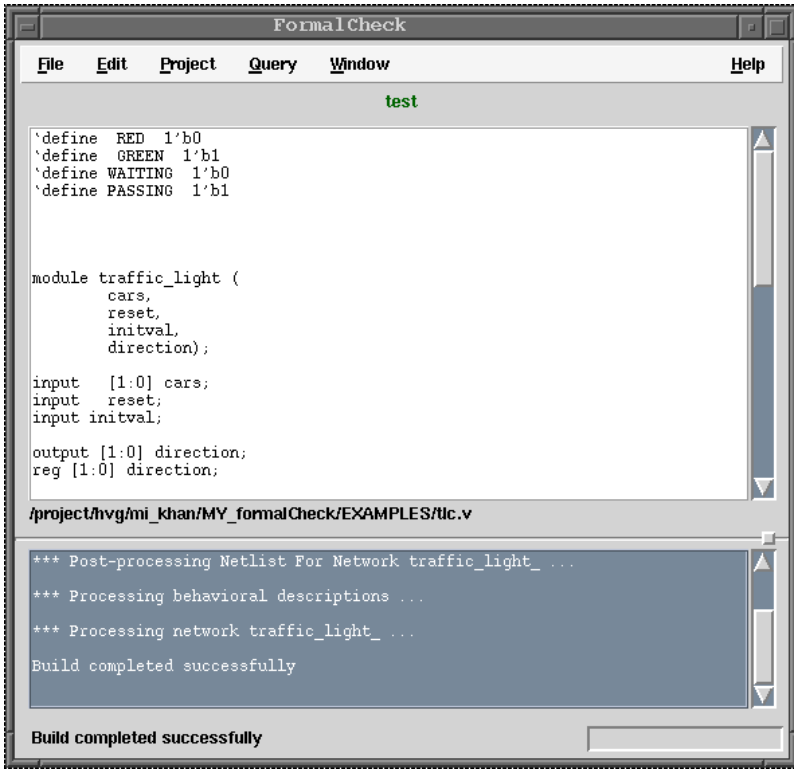


FIG 3.2: Running the Build option on an example (for *Step 4*)

Step 4 (Compiling the Verilog/VHDL design - BUILD).

PROJECT

BUILD

"Build" option is the first main step towards verification. Whenever any change in the design is made, BUILD is necessary, so as FILE-SAVE.

The following three actions are performed implicitly by "Build":

i) Compilation of the design (Verilog or VHDL)

ii) Post-Processing of the netlist.

iii) Processing of the behavioral description.

Compilation errors are highlighted in the Diagnostic region, and then are hyper-links. Clicking on an error will highlight the source line in the edit region that caused the error.

Note: At the end of this Step 4, the project has to be saved once more (repetition of Step 2

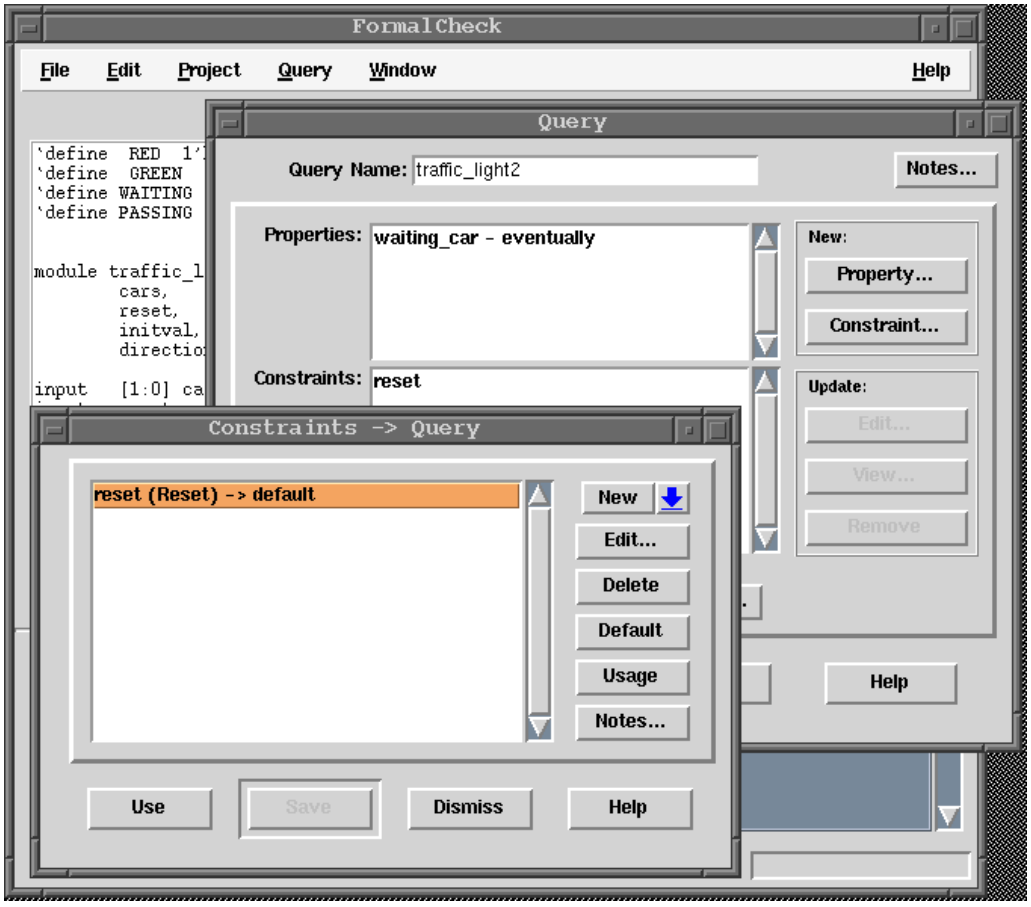


FIG 3.3: The Constraint Library Panel (for *Step 5*)

Step 5 (Adding a Constraint).

PROJECT

CONSTRAINT LIBRARY

This will invoke "**Constraint Library**" panel (Figure 3.3). Constraints limits the state place of a design model that is verified [6].

Note: Please refer to the previous chapter for more details on CONSTRAINTS.

Step 6 (Create/Edit Query).

QUERY

NEW / EDIT

A query must contain one or more properties. The procedure to write properties are explained in the previous chapter. A new query also automatically contains any constraints marked as default in the constraint library. Additional constraints can be added or removed manually.

Default Constraint: "A default constraint is a constraint that will be automatically applied to all new queries in a project. A constraint marked as default will not be applied to any pre-existing queries unless specified manually. A non-default constraint can be manually added to any query" [6].

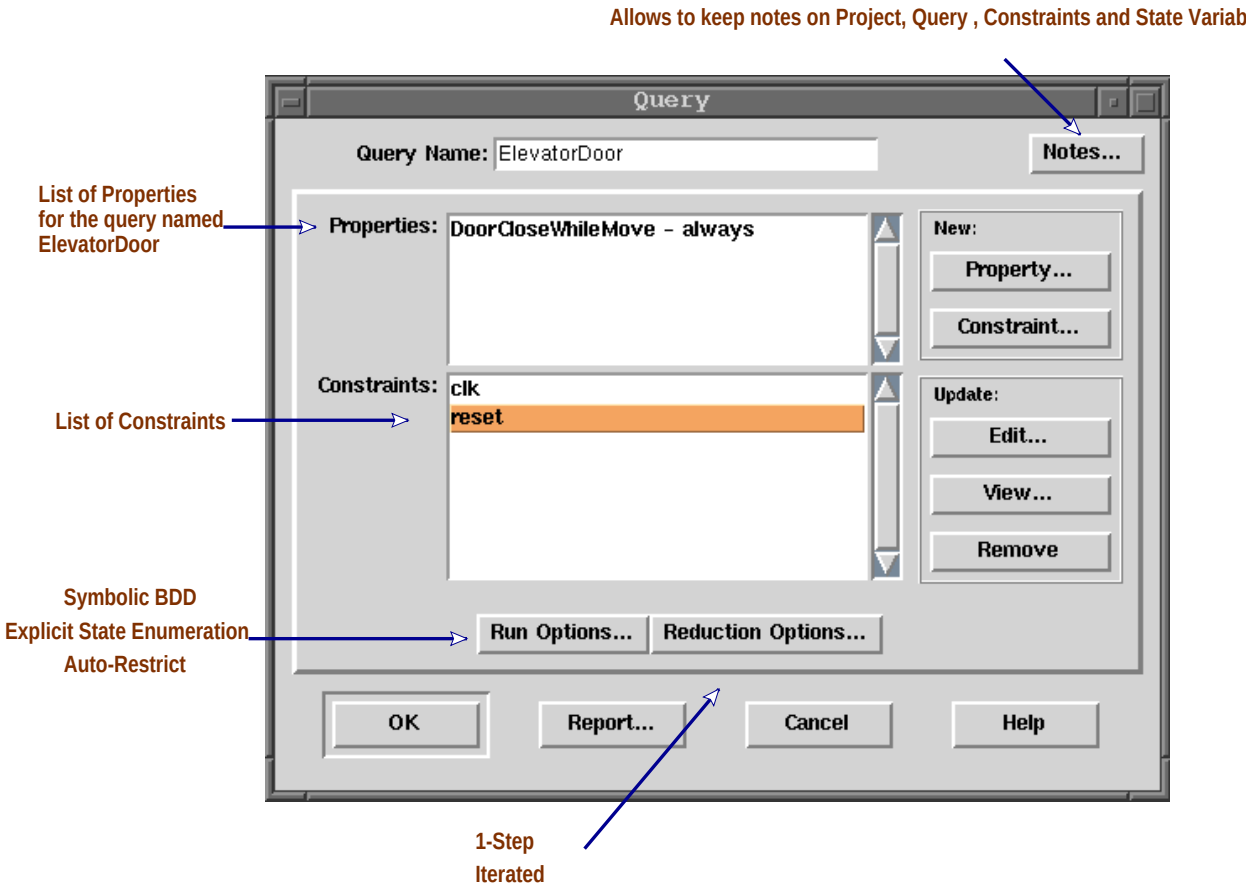


FIG 3.4: The "Query" Panel and Its Features (for *Step 6*)

Four main steps in the “Query” panel to remember:

1. Name of the property.
2. Type of property.
3. Insertion of logic in Enabling Condition, Fulfilling Condition and/or Discharging Condition.
4. Taking care of Constraints.

“Fulfill Delay”: A delay can be added between the Enabling Condition and the checking of the Fulfilling Condition. This delay is specified as an integer that counts the occurrences of an event which is again specified by a boolean expression written into the “**Of Edge**” field (please refer to Figure 3.3, “Property” panel).

“Duration”: The verification windows terminates after a given duration of the Discharging condition becomes true, whichever comes first. The duration is measured by an integer, same way as the “Fulfill Delay”.

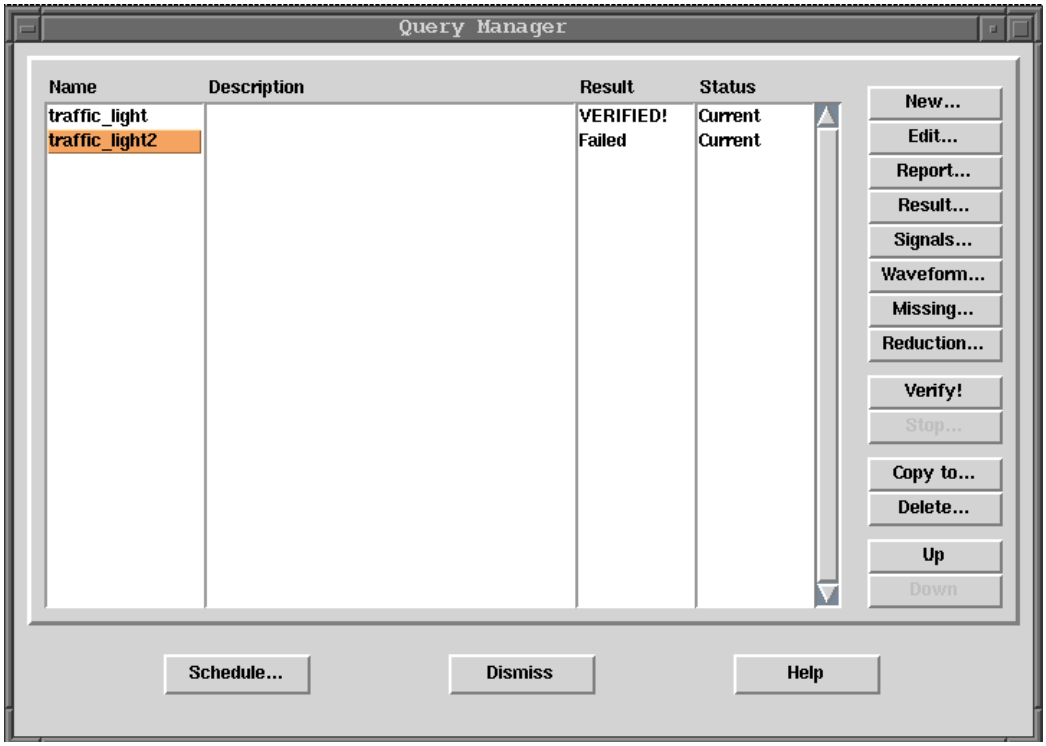


FIG 3.5: The “Query Manager” panel (for *Step 7*).

Step 7 (Verify Query).

Before starting to verify the query, the user has to save the query as well as the project (Step 2).

QUERY

QUERY MANAGER

Verify!

The user has to select the desired query from the “Name” column (Figure 3.5) and press the “**verify!**” button. FormalCheck will do the rest about verification.

4. Post-Processing Analysis

The "**Result**" column of the "Query Manager" may contain eight possible outcomes. The following is the short description:

(1) New: Indicates a new query, no attempt has been taken to verify it.
(2) Failed: Error has been found in the design and an error track is available for debugging.
(3) Verified!: The query was previously verified.
(4) Running: A verification procedure is currently being processed.
(5) Terminated: The verification process is incomplete (user or some other intervention).
(6) Scheduled: The verification is scheduled by the "Schedule Manager" (refer to the Query pull down menu).
(7) No Error: No failures have been found, but this is not same as verification. It sometimes happens while using the auto-restrict algorithm to quickly verify the design.
(8) Vacuous: The Enabling condition is never satisfied, the reason why the Fulfilling condition was never checked.

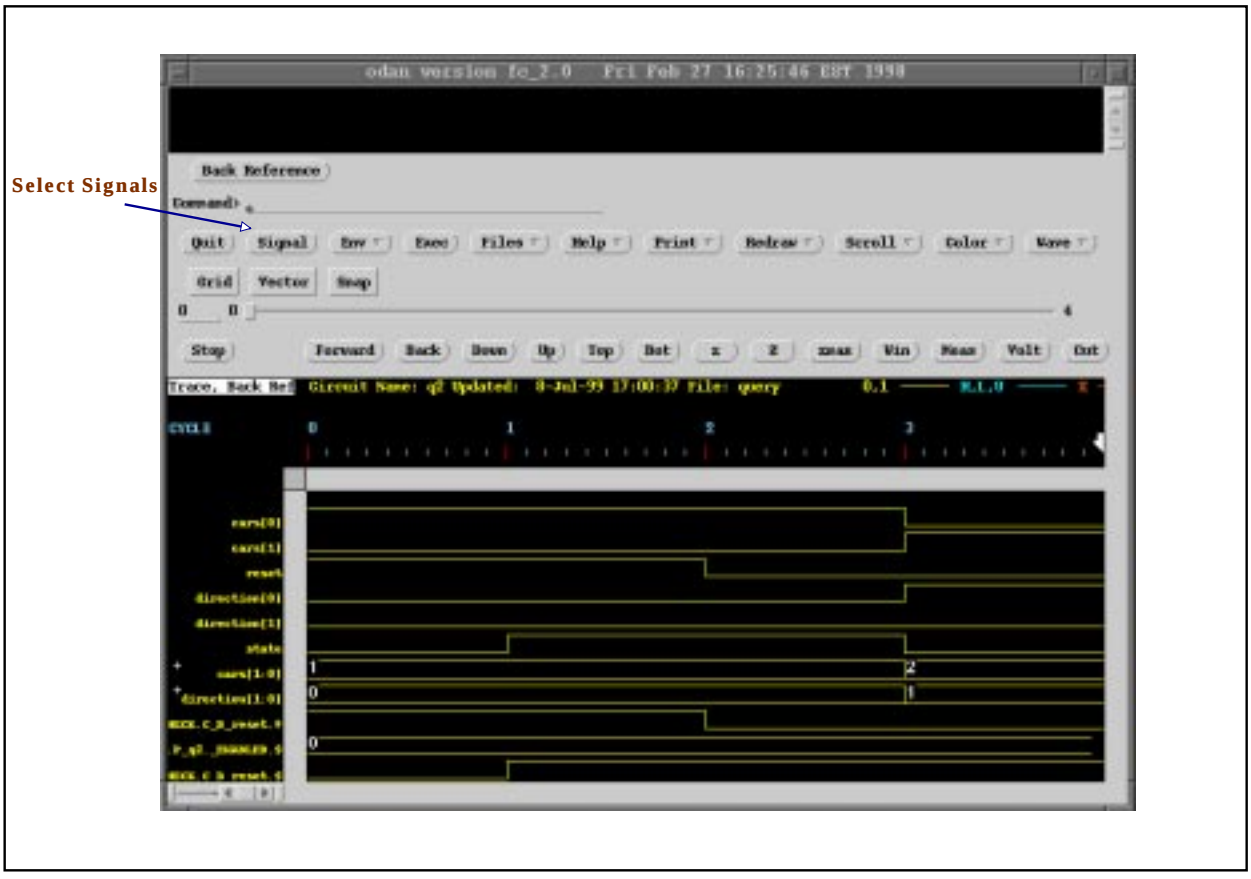


FIG 4.1: The Wave Form Window (ODAN).

If the result is "Failed", then the waveform viewer (the Output Display / Analysis Tool - ODAN) is very useful to finish the post-verification analysis. FormalCheck creates 2 files

(.HDR and .REC) in the query directory for each failed query and ODAN uses those files to offer the following capabilities:

- Interactively select a subset to the signals for display.
- Zoom-in on critical areas.
- Select a specific time region to view signal activity in detail.
- Group signals under one label.

If the result is “Vacuous”, then there may be a possibility that it was over-constraint (please refer to Chapter 2 of this manual).

5. Advanced Options

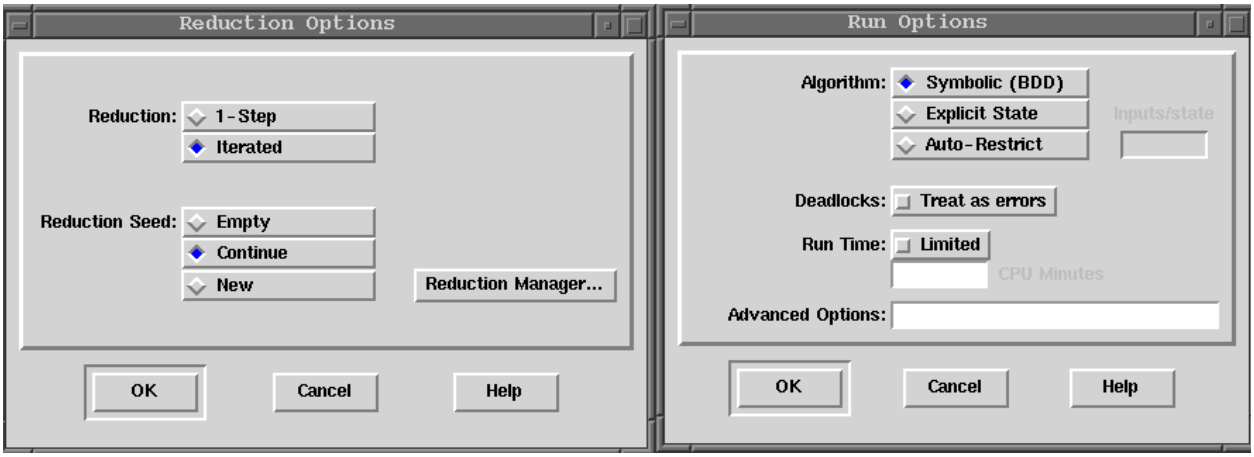


FIG 5.1: Reduction Options panel & Run Options panel

FormalCheck uses a patented localized reduction algorithm to reduce the size of the design model relative to the property being tested [6]. Among the two algorithms **1-step** is used by default and the **iterated algorithm** is used for complex designs, because it takes *less memory* (more time) to verify the design. This reduction technique finds a portion of the design model on which it is sufficient to run the verification.

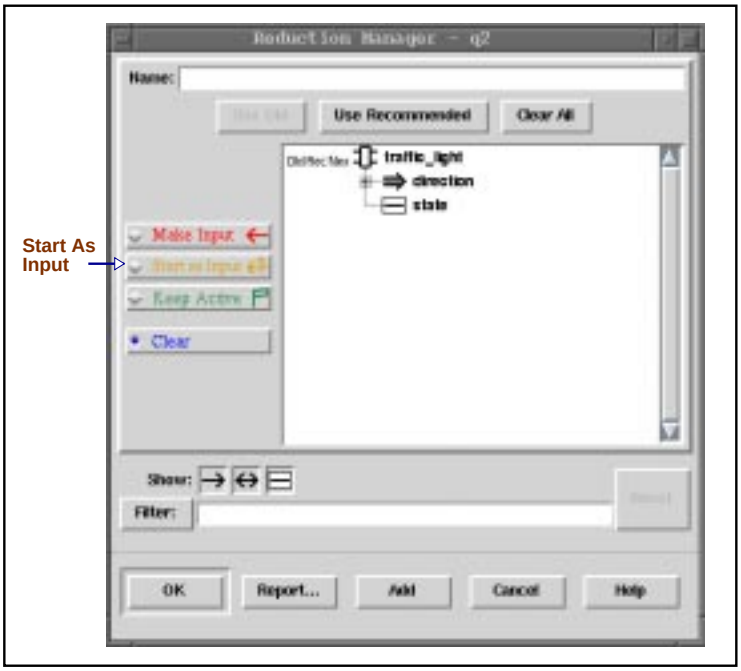


FIG 5.2: Reduction Manager panel

A r**eduction seed** can be provided by the user to further speed up the verification process. The user designates a candidate portion of the design model by marking its boundary signals **Start As Input**. This option disconnects all signals marked as "Start As Input" from the driving expressions and turns them into primary inputs. If some driving expressions really in need of some of these signals, the algorithm will reinstate them, cutting the design model signals at an alternate point instead [6].

Note: Iterated reduction does not support Auto-Restrict verification algorithm
Reduction Seed is recommended when Iterated Algorithm is used.

Through the "**Make Input**" option some signals can be disconnected almost permanently from their driving expressions, and thus treated as primary inputs. This is the way to isolate a portion of the design model. In such a condition, if FormalCheck verifies a query, it means that the same query would also be verified in the full design model.

Note: "Make Input" sometimes creates problems in error tracing if FormalCheck returns failure.

Through the option "**Keep Active**", one can make sure that the signal is not going to be excluded from the verification, no matter what.

Note: If "Keep Alive" is used, it increases the computational cost.

"Clear" option removes any reduction designation from a design element.

New Features:

AutoCheck: AutoCheck is applied as a preverification procedure. It is used in the early design stages to quickly run on queries in order to find early design bugs. AutoCheck is not a complete verification procedure as it does not cover the entire state space. (further reference: page 5-9 in user's manual: command "formalreadthemanual").

Macro: Macro is the shorthand version of any expression. It is done by the expression editor. The procedure is to type or insert any expression and give it a name after pressing "MakeMacro".

Clock Extraction: While imposing clock constraints, the feature "Clock Extraction" can be used to reduce verification time in some cases. This option should be tried if the design is synchronous with respect to a single clock edge.

Script Interface: FormalCheck supports user-defined UNIX script to perform tasks, like automatically generating constraints. (see page 8-7 in users' manual).

IMPORTANT BUILD OPTIONS:

[-D <parameter>=<value>]
Allows the value of a Verilog parameter to be set to a different (smaller) value to reduce the state space. This is necessary for parenthesized RAMs, ROMs, FIFOs, and queues.

[-W<path>]
Specifies a single directory to which all the files will be created by FormalCheck.

ADVANCED QUERY RUN OPTIONS:

[-L<mb>]
Specifies a limit on memory used. “mb” is memory in megabyte, represented in floating point notation. 80% limit of user accessible memory is recommended. The verification begins a wrap up sequence (time consuming) when the memory is finished to generate reports.

[-#hardlimit=<mb>]
Stops verification without the warp up sequence.

[-DVACCHECK]
This option verifies if really a query is **vacuous**. If the result with this option is “verified!” then the query is vacuous. If a bad cycle is found then the query is not vacuous.

THREE VERIFICATION ALGORITHMS:

- Symbolic State Enumeration (Ordered Binary Decision Diagrams / Symbolic BDD)
- Explicit State Enumeration.
- Auto-Restrict.

Symbolic BDD is generally useful for verifying models with a very large set of States. But, BDDs become extremely large for the design models with arithmetic expressions.

Explicit State Enumeration is recommended for designs with fewer than 1000 primary inputs/state and which contain a large number of arithmetic expressions. It can find error in the design faster than the symbolic BDD. This algorithm is preferable for liveness property.

Auto-Restrict attempts to narrow down the portion of the design where a failure is likely to be found. It speeds up the verification. But if may give the result “No Errors”. In this case, the verification has to be performed again with any of the previous two algorithms.

A Recommended Procedure for Verification:

At first Symbolic BDD should be used with a time limit. The verification can stop for three reasons. An error can be found, the time limit can be expired or the verification can be complete. For the first two reasons, one can do the following:

- Missing value report should be checked.
- Restriction should be added.
- Reduction Manager should be employed.
- Verification should be re-run with Auto-restrict or Explicit State Enumeration.

Note: "Missing Values": Show the design elements (signals) that are irrelevant to the specific query

References

[1] A.J. Hu. Formal Hardware Verification with BDDs: An Introduction, IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing (PACRIM), pp. 677-682, 1997.

[2] E.M. Clarke and J.M. Wing. Formal Methods: State of Art and Future Directions, CMU Computer Science Technical Report CMU-CS-96-178, August 1996.

[3] R.P. Kurshan. Formal Verification In a Commercial Setting, Bell laboratories, Murry Hill, NJ. In Design Automation Conference. June, 1997.

[4] R.H. Hardin, Z. Har'EL, R. P. Kurshan, COSPAN, Springer LNCS 1102 (1996), pp. 61-67.

[5] R. P. Kurshan, Computer-Aided Verification of Coordinating Processes, Princeton Univ. Press, 1994.

[6] FormalCheck User's Guide. Cadence Design Systems. V2.3, August 1999.

APPENDIX A

System Settings to Run FormalCheck

To use the FormalCheck tool from Cadence:

1) type from the UNIX prompt:
source /CMC/ENVIRONMENT/formalcheck.env
(this is actually a symbolic link to /**CMC/ENVIRONMENT/formalcheck_2004a.env**, if you are sourcing a copy of the .env from your home directory MAKE SURE YOU COPY THE NEWEST VERSION of the **/CMC/ENVIRONMENT/formalcheck.env** to your directory)

2) to start the tool type:
formalcheck -gui (to start the Graphical User Interface version)
formalcheck (to start the command line version)

3) to read the on-line documentation:
in the Main Graphical User Interface window, simply click on the Help button in the upper left-hand side, a pop-up list will be presented:
Formalcheck Reference Manual
Formalcheck Gui Reference Guide
Related Products
Whats New
About Formalcheck

Selecting any of these will start the Cadence documentation by invoking the Netscape browser and opening up the selected HTML documents.

For those users who prefer to read PDF format, there is a 343 page manual in:
/**CMC/tools/cadence/LDV/doc/fcheck/fcheck.pdf**

4) There is a small introductory example and short tutorial to using Formalcheck in the directory:
/**CMC/tools/cadence/LDV/tools/fcheck/examples/chap1_example**
This directory contains:

```
drwxr-xr-x   4 cmcmgr   cmcmgr       512 May 29  2001 .
   drwxr-xr-x  11 cmcmgr   cmcmgr       512 May 29  2001 ..
   -rwxr-xr-x   1 cmcmgr   cmcmgr    163679 May 29  2001 overview.pdf
   drwxr-xr-x   2 cmcmgr   cmcmgr       512 May 29  2001 vhdl
   drwxr-xr-x   3 cmcmgr   cmcmgr       512 May 29  2001 vlog
```

The overview.pdf file is Chapter 1 of the fcheck.pdf document.

The directories vhdl and vlog contain the source code for the example used in the introductory example.

APPENDIX B
FormalCheck Files Description

The following is the list of files created in the directory called .<priject_name> (the name used for the project.

DIRECTORY .crd_project.fpj

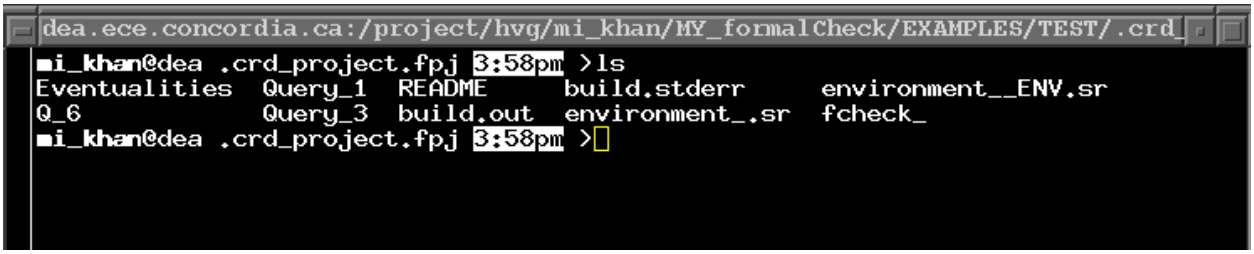


FIG 1: SnapShot of the Directory .crd_project.fp

Contents of the file README

All the file contained in this directory and below are considered part of the FormalCheck project. Changing anything in this directory or its children could result in corrupting your verification results.

DIRECTORY Query_1

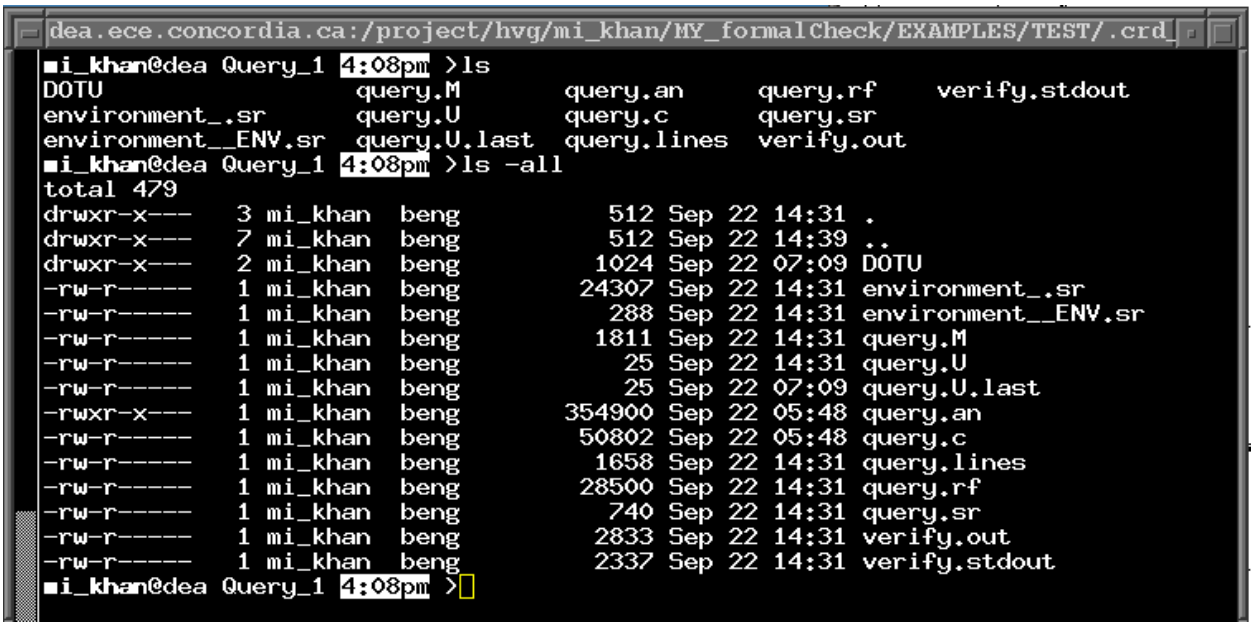

dea.ece.concordia.ca:/project/hvg/mi_khan/MY_formalCheck/EXAMPLES/TEST/.crd
mi_khan@dea Query_1 4:08pm >ls
DOTU query.M query.an query.rf verify.stdout
environment_.sr query.U query.c query.sr
environment__ENV.sr query.U.last query.lines verify.out
mi_khan@dea Query_1 4:08pm >ls -all
total 479
drwxr-x--- 3 mi_khan beng 512 Sep 22 14:31 .
drwxr-x--- 7 mi_khan beng 512 Sep 22 14:39 ..
drwxr-x--- 2 mi_khan beng 1024 Sep 22 07:09 DOTU
-rw-r----- 1 mi_khan beng 24307 Sep 22 14:31 environment_.sr
-rw-r----- 1 mi_khan beng 288 Sep 22 14:31 environment__ENV.sr
-rw-r----- 1 mi_khan beng 1811 Sep 22 14:31 query.M
-rw-r----- 1 mi_khan beng 25 Sep 22 14:31 query.U
-rw-r----- 1 mi_khan beng 25 Sep 22 07:09 query.U.last
-rwxr-x--- 1 mi_khan beng 354900 Sep 22 05:48 query.an
-rw-r----- 1 mi_khan beng 50802 Sep 22 05:48 query.c
-rw-r----- 1 mi_khan beng 1658 Sep 22 14:31 query.lines
-rw-r----- 1 mi_khan beng 28500 Sep 22 14:31 query.rf
-rw-r----- 1 mi_khan beng 740 Sep 22 14:31 query.sr
-rw-r----- 1 mi_khan beng 2833 Sep 22 14:31 verify.out
-rw-r----- 1 mi_khan beng 2337 Sep 22 14:31 verify.stdout
mi_khan@dea Query_1 4:08pm >

FIG 2: SnapShot of the Directory Query_1

Contents of the file verify.stdout

model M: FCHECK bcy0
model unM: FCHECK.C_B_clk FCHECK.C_B_reset
model M_p:
model M_o: .environment_._H15_road_B_._VD5_state___0__1__[1]
M_o+: .environment_._H15_road_B_._VD5_state___0__1__[0]
M_o+: .environment_._H14_road_A_._VD5_state___0__1__[1]
M_o+: .environment_._H14_road_A_._VD5_state___0__1__[0]
pruned:
.environment__ENV
.environment_.__fi_status_A_[0]
.environment_.__fi_status_A_[1]
.environment_.status_A_[0]
.environment_.status_A_[1]
.environment_.__fi_test_
.environment_.test_
.environment_.__fi__VD0__T161___0__2__[0]
.environment_.__fi__VD0__T161___0__2__[1]
.environment_.__fi__VD0__T161___0__2__[2]
.environment_._VD0__T161___0__2__[0]
.environment_._VD0__T161___0__2__[1]
.environment_._VD0__T161___0__2__[2]
.environment_._H13_police_
.environment_._H14_road_A_.__fi_status_[0]

.environment_._H14_road_A_.__fi_status_[1]
.environment_._H14_road_A_.status_[0]
.environment_._H14_road_A_.status_[1]
.environment_._H14_road_A_.__fi__VD5_state___0__1__[0]
.environment_._H14_road_A_.__fi__VD5_state___0__1__[1]
.environment_._H14_road_A_._H4__I3
.environment_._H14_road_A_._H5__I4
.environment_._H14_road_A_._H6__I5
.environment_._H15_road_B_.__fi_status_[0]
.environment_._H15_road_B_.__fi_status_[1]
.environment_._H15_road_B_.status_[0]
.environment_._H15_road_B_.status_[1]
.environment_._H15_road_B_.__fi__VD5_state___0__1__[0]
.environment_._H15_road_B_.__fi__VD5_state___0__1__[1]
.environment_._H15_road_B_._H4__I3
.environment_._H15_road_B_._H5__I4
.environment_._H15_road_B_._H6__I5
.environment_._H16_col_
.environment_._H17_starv_
.environment_._H18__I12
.environment_._H19__I13
.environment_._H20__I14
.environment_._H21__I15
.FCHECK.C_B_reset
active:
.FCHECK.P_collision
.bcy0
free:
.environment_._H14_road_A_._VD5_state___0__1__[0]1 free choices
.environment_._H14_road_A_._VD5_state___0__1__[1]1 free choices
.environment_._H15_road_B_._VD5_state___0__1__[0]1 free choices
.environment_._H15_road_B_._VD5_state___0__1__[1]1 free choices

product of free choices=1

free:
.environment_1x1x1x1=1 free choices

product of free choices=1

looking for:
400486bde2e4888a96a0765ca3de8bc06f3d7426b004e4b0394012c88d522073c4344bc5dd17fdb12a43dd391da3ae8bd3b78a0eb6ad72a55f955107
FCkillMonitor 9771: PIDS is 9773

Contents of the file verify.out

9771
Verification Server: enterprise.ece.concordia.ca 9771
SunOS enterprise.ece.concordia.ca 5.5.1 Generic sun4u sparc
cospan -I/CMC/tools/formalcheck2.1/SOLARIS/include environment_.sr -Ks -#caseonedflt -#varlines -#nmi -#missingasgn -#nocaseonedflterr -#shortfloat=3 -#status -#dupstvars -#hotunroot -#hotback -#Msuperset -b -q -#csplit -#pthreshold=1e3,50 -#rmc -#flow -#disconnect -#slowdisconnect=4 query.sr
cospan: Version 8.23.24 (Bell Laboratories) 27 May 1998
Iterative run for option -q
Iteration 0:
++++++++++++++++++
cospan: Version 8.23.24 (Bell Laboratories) 27 May 1998
+ sr_E -I/CMC/tools/formalcheck2.1/SOLARIS/cospan -#caseonedflt -#varlines -#nmi -#missingasgn -#nocaseonedflterr -#shortfloat=3 -#status -#dupstvars -#hotunroot -#hotback -#Msuperset -#pthreshold=1e3,50 -#flow -#disconnect -#slowdisconnect=4 -I/CMC/tools/formalcheck2.1/SOLARIS/cospan -I/CMC/tools/formalcheck2.1/SOLARIS/include environment_.sr -Ks -#caseonedflt -#varlines -#nmi -#missingasgn -#nocaseonedflterr -#shortfloat=3 -#status -#dupstvars -#hotunroot -#hotback -#Msuperset -b -#reduction -#pthreshold=1e3,50 -#flow -#disconnect -#slowdisconnect=4 query.sr -#caseonedflt -#varlines -#nmi -#missingasgn -#nocaseonedflterr -#shortfloat=3 -#status -#dupstvars -#hotunroot -#hotback -#Msuperset -#pthreshold=1e3,50 -#flow -#disconnect -#slowdisconnect=4 -#caseonedflt -#varlines -#nmi -#missingasgn -#nocaseonedflterr -#shortfloat=3 -#status -#dupstvars -#hotunroot -#hotback -#Msuperset -#reduction -#pthreshold=1e3,50 -#flow -#disconnect -#slowdisconnect=4
Status: Begin parsing at 0.01 sec 0 megabytes.
query.rf: Wed Sep 22 14:31:28 1999
environment_.sr: Wed Sep 22 14:31:31 1999
./environment__ENV.sr: Wed Sep 22 14:31:31 1999
query.sr: Wed Sep 22 14:31:29 1999
/CMC/tools/formalcheck2.1/SOLARIS/include/QRY.h: Wed Jul 29 14:17:28 1998
/CMC/tools/formalcheck2.1/SOLARIS/include/QRY+.h: Wed Jul 29 14:17:27 1998
/CMC/tools/formalcheck2.1/SOLARIS/include/gui.h: Wed Jul 29 14:17:28 1998
Status: Begin checks and tree rewrites at 0.15 sec 1.37626 megabytes.
query.rf: list entry count: unM 2 M 2 M_o 4
568 pruned, 1 active, 4 freed by reduction
284 data variables declared or with width >= -#databits=4
4 selection/local variables
1 bounded state variables: 2 states
0 unbounded state variables
1 boolean cysets
0 boolean recurs
0 free selection/local variables: 1 selections/state
2 kill/free optimization actions
2 variable assignments driven by kills
886 variable reference clippings, 655 expression clippings
2 vector bitwise comparisons expanded
0 pausing processes
0 non-deterministic (non-free) selection/local variables
1 selections/state (maximum)
1 total selections/state (maximum)

sr_E: Equivalent reduction, Task performed! (older run)
+ exit 3
FormalCheck Verification Finished
Wed Sep 22 14:31:53 EDT 1999

Contents of the file query.c

```
static char WHAT[]="@(#)query.c Wed Sep 22 05:49:05 1999";
#ifndef Int
#define Int int
#endif
#define String int
#define BYTESIZE 8
#ifdef HDR
extern
#endif
struct{
struct{
struct{
Int _clk_;
unsigned char U_clk_[2];
Int _reset_;
unsigned char U_reset_[2];
struct{
Int _prev0;
}_91179_clk____;
}_environment__ENV;
struct{
Int ___fi__U2_r_state_;
Int ___fi__U3_r_state_;
struct{
Int _9983___fi__VD5_state___0__1___0;
Int _9982___fi__VD5_state___0__1___1;
Int _9985__VD5_state___0__1___0;
Int _9984__VD5_state___0__1___1;
struct{
Int _9923___fi_SR_OUT_state__0;
Int _9922___fi_SR_OUT_state__1;
Int _9925_SR_OUT_state__0;
Int _9924_SR_OUT_state__1;
Int _9911___fi_SR_ST_state__0;
Int _9910___fi_SR_ST_state__1;
struct{
Int _st0;
}_9913_SR_ST_state__0;
struct{
Int _st1;
}_9912_SR_ST_state__1;
Int ___fi__T90_;
Int __T90_;
unsigned char U__T90_[2];
Int _9926___fi__Z1_state__1;
Int _9928__Z1_state__1;
}__H6__I5;
}__H14_road_A_;
struct{
```

..... FILE CONTINUES

APPENDIX C

Design Tips (for verification purpose)

One may remember the following:

- One should not use two signals to activate the events in an always block.

  correct: `always @(posedge clk)`

  incorrect: `always @(posedge clk && reset)`

- "Initialization" is not supported by FormalCheck. We used another extra signal "reset", propagated through all the modules. The "reset" signal is made active high for only 2 cranks of time and low for ever (Reset/Repeat constraint). The always block activated by the positive "reset" signal contains the assignments for the initialization.

Note that while specifying the above reset signal one is going to introduce another "always" block (concurrent block). With two always blocks in the same module, one should be careful to avoid reassignment of the same signal in the 2 concurrent blocks. This may greatly affect the verification.

- If design contains internal nondeterministic signals, they should be converted to primary input of that module because, FormalCheck does not support nondeterministic internal signals.

- FormalCheck does not support real enumerated data types for Verilog code. But it is possible to use the key word "define". For example,

```
‘define no_cars  0
‘define car_waiting  1
‘define cars_passing  2
‘define traffic_status {no_cars, car_waiting, cars_passing}
```

Note that the above piece of Verilog code is used just outside the modules (please refer to the Verilog example in Appendix D).

APPENDIX D

The "Arbiter" Example

(Courtesy of Cadence for educational purposes only)

==========

Description of the Arbiter to be verified:

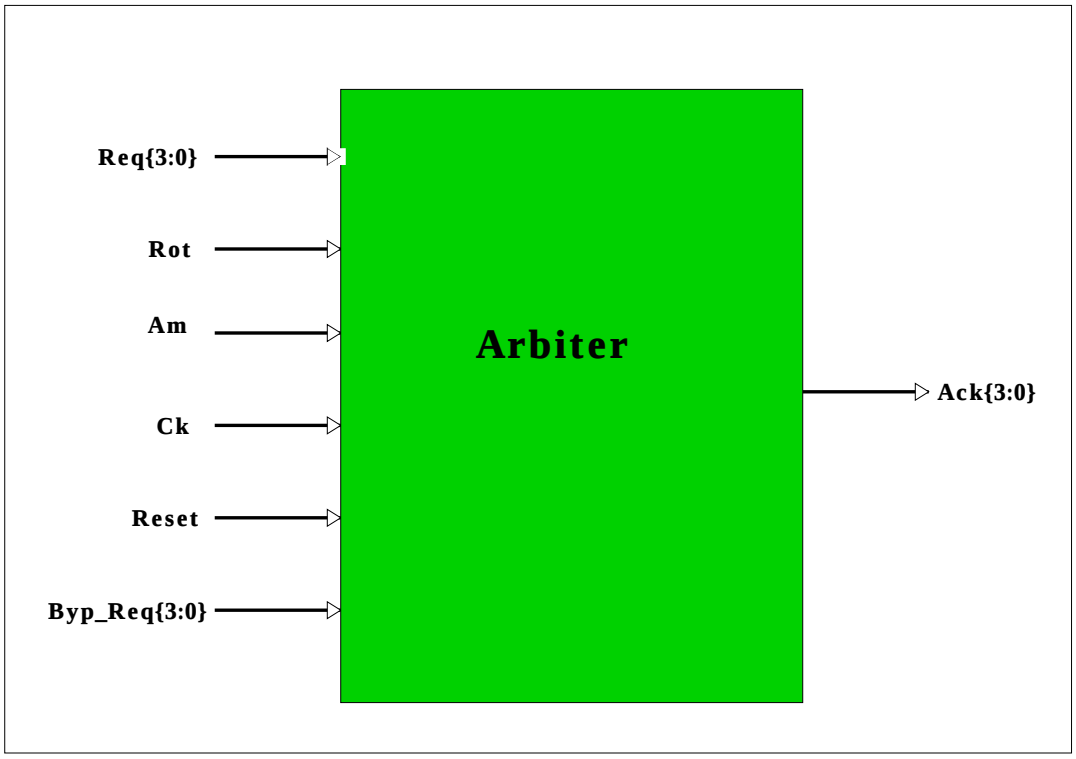


FIG 3: Arbiter Circuit Block Diagram

The circuit arbitrates a shared resource among 4 clients. It features a selectable (clockwise or counter clockwise) polling direction for either a round-robin or an aged-based arbitration scheme. The circuit also features the ability to bypass requests. The above picture shows the block diagram of the circuit.

The state diagram of the circuit's functionality is depicted on the next page.

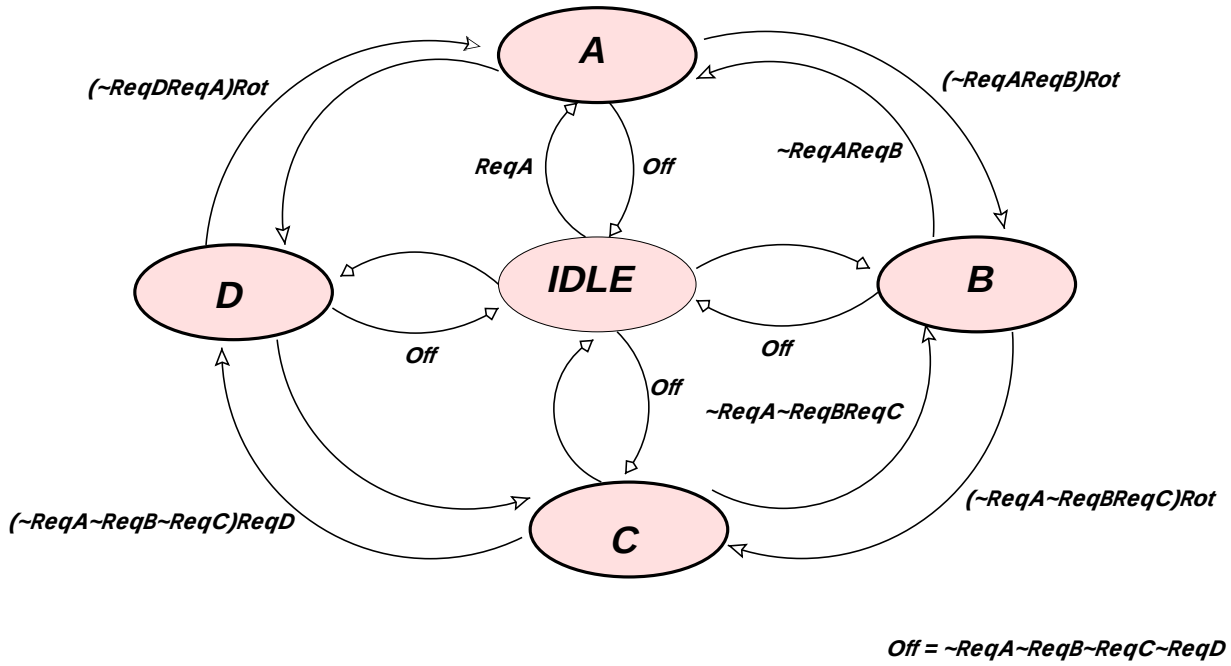


FIG 4: Arbiter State Diagram

Table 1: Arbiter's I/O Specification

SIGNAL	POLARITY	BRIEF DESCRIPTION
Req[3:0]	1	Client's Request - Active High
Byp_Req[3:0]	1	Command to Bypass Requests Active High.
Rot	0/1	0 - Counter -clockwise Rotation 1 - Clockwise Rotation
Am	0/1	Arbitration Mode 0 - Round Robin 1 - Age Based
Ck	Posedge	System Clock
Reset	1	Reset Signal - Active High
Ack[3:0]	1	Acknowledgement - Active High

Arbiter Basic Specification:

- Request inputs (Req[3:0]) are required to be deasserted a minimum of one clock cycle after the request is acknpwledged.

- Acknowledge outputs (Ack[3:0]) will remain stable one clock cycle after the corresponding Request (Req[3:0]) input goes low.

The following are all the necessary verilog files of the RTL design:

==========

RTL design:

arb.h

```
//  ********************************************************************************
//
//File:arb.h
// Description:Global Mnemonic Assignments in arbiter
//
//  ********************************************************************************
//

‘define IDLE4’b0000
‘define GrantA4’b0001
‘define GrantB4’b0010
‘define GrantC4’b0100
‘define GrantD4’b1000
```

arb.v

```
//  ************************************************
//
//File:arb.v
// Description:Top RTL-Level for arbiter
//
//  ************************************************
//
//
//  ************************************************
//
//    circuit:arb.v
//  Description:Arbiter for 4 clients.
//
//  ************************************************
//
//
// Files:
//arb.h   --‘include "arb.h"
```

```
//dpath.v -- Decision logic in a small datapath.
//fsm.v   -- FSM for arbiter
//age.v   -- State to compute the age of a client's request
//arb.v   -- Top RTL-level for arbiter
//arb.fpj -- Project file (arb.fpj.ascii --> text mode project file)
//
//
// ********************************************************************
//
//Brief Functional Description
//
// ********************************************************************
//
// Arb.v arbitrates a shared resource among 4 clients.
//
// A Request "Req_(j)" is eventually acknowledged. Asserting "Ack_(j)"
// after a request is issued may take as little as 1 clock cycle or as
// many as "X" clock cycles. It is expected from the environment that
// each Req_(j) asserted is eventually withdrawn. Removing Req_(j)
// after the acknowledment is issued may take as little as 1 clock cycle
// or as many as "Y" clock cycles. Each Ack_(j) goes away 1 clock cycle
// after Req_(j) is withdrawn.
//
//



modulearb (ck, reset, req, byp_req, rot, am, ack);
input      ck, reset;
input[3:0] req, byp_req;
input      rot, am;
output[3:0] ack;


// ********************************************************************
//
//Brief Description on I/O signals
//
// ********************************************************************
//
// Inputs
// -------
// cksystem clock
// resetsynchronous reset - Active high
// reqbus requests from clients
// byp_reqcommand to bypass requests from clients
// rotarbiter's rotation direction
//-----------+---------------------
// rot Value | Polling Direction
//-----------+---------------------
//     0     | CounterClockwise
//     1     | Clockwise
//
// amarbitration mode (scheme)
//-----------+---------------------
//  am Value | Arbitration Scheme
//-----------+---------------------
//     0     | Round Robin
//     1     | Aged-based
```

```
//
// Outputs
// -------
// ackacknowledgement of bus grants back to clients
//
// ***************************************************************



// ***************************************************************
//  Internal wires
// ***************************************************************
wireselA, selB, selC, selD, aged;
wire[2:0]AgeA, AgeB, AgeC, AgeD;
wire[3:0]requests, ack;
wirereqA, reqB, reqC, reqD;

  assign aged = am;
  assign reqA = ~ byp_req[0] & req[0];
  assign reqB = ~ byp_req[1] & req[1];
  assign reqC = ~ byp_req[2] & req[2];
  assign reqD = ~ byp_req[3] & req[3];
  assign requests = { {reqD},  {reqC}, {reqB}, {reqA} };

fsm  FSM (.ck(ck), .reset(reset), .reqA(selA), .reqB(selB),
  .reqC(selC), .reqD(selD), .state(ack));

dpath DP (.AgeA(AgeA), .AgeB(AgeB), .AgeC(AgeC), .AgeD(AgeD),
  .rot(rot), .aged(aged), .req(requests), .lastGrant(ack),
  .selA(selA), .selB(selB), .selC(selC), .selD(selD) );

age  AGE (.ck(ck), .reset(reset), .req(requests), .ack(ack),
  .AgeA(AgeA), .AgeB(AgeB), .AgeC(AgeC), .AgeD(AgeD));

endmodule // arb
```

fsm.v

```
//   ********************************************************************************
//
//File:fsm.v
// Description:Arbiter's Finite State Machine
//
//   ********************************************************************************
modulefsm (ck, reset, reqA, reqB, reqC, reqD, state);

inputck, reset, reqA, reqB, reqC, reqD;
output[3:0] state;

reg[3:0] state;

  always @(posedge ck)
  begin
     if (reset)
state <= 'IDLE;
```

```
    else
    begin
case (state)
  'IDLE ://Waiting for Requests
    begin
if ( (reqA | reqB | reqC | reqD)  == 1'b1)
  begin
if (reqA)
state <= 'GrantA;
  else if (reqB)
state <= 'GrantB;
  else if (reqC)
state <= 'GrantC;
  else if (reqD)
state <= 'GrantD;
  end
    end

  'GrantA :
    begin
if ( (reqA | reqB | reqC | reqD)  == 1'b0)
state <= 'IDLE;
else
  begin
if (reqB)
state <= 'GrantB;
  else if (reqC)
state <= 'GrantC;
  else if (reqD)
state <= 'GrantD;
  end
    end

  'GrantB :
    begin
if ( (reqA | reqB | reqC | reqD)  == 1'b0)
state <= 'IDLE;
else
  begin
if (reqA)
state <= 'GrantA;
  else if (reqC)
state <= 'GrantC;
  else if (reqD)
state <= 'GrantD;
  end
    end

  'GrantC :
    begin
if ( (reqA | reqB | reqC | reqD)  == 1'b0)
state <= 'IDLE;
else
  begin
if (reqA)
state <= 'GrantA;
  else if (reqB)
state <= 'GrantB;
  else if (reqD)
```

```
state <= ‘GrantD;
   end
     end

  ‘GrantD :
     begin
if ( (reqA | reqB | reqC | reqD)  == 1’b0)
state <= ‘IDLE;
else
   begin
if (reqA)
state <= ‘GrantA;
   else if (reqB)
state <= ‘GrantB;
   else if (reqC)
state <= ‘GrantC;
   end
     end
endcase

     end

 end

endmodule // fsm
```

dpath.v

```
//   ********************************************************************************
//
//File:dpath.v
// Description:Datapath with Grant’s Selection Logic
//
//   ********************************************************************************
//


‘include "arb.h"

module dpath (AgeA, AgeB, AgeC, AgeD, rot, aged, req,
            lastGrant, selA, selB, selC, selD);

input[2:0]AgeA, AgeB, AgeC, AgeD;
inputrot, aged;
input[3:0]req, lastGrant;
outputselA, selB, selC, selD;

//
//   ****************************************************************
//
//Internal wires
//
//   ****************************************************************
//

wire A_ge_B, A_lt_B, A_ge_C, A_lt_C, A_ge_D, A_lt_D;
wire B_ge_A, B_lt_A, B_ge_C, B_lt_C, B_ge_D, B_lt_D;
```

```
wire C_ge_A, C_lt_A, C_ge_B, C_lt_B, C_ge_D, C_lt_D;
wire D_ge_A, D_lt_A, D_ge_B, D_lt_B, D_ge_C, D_lt_C;
wire selA_pri0, selA_pri1, selA_pri2, selA_pri3;
wire selB_pri0, selB_pri1, selB_pri2, selB_pri3;
wire selC_pri0, selC_pri1, selC_pri2, selC_pri3;
wire selD_pri0, selD_pri1, selD_pri2, selD_pri3;
wire selA_pri2_1, selA_pri2_2, selA_pri2_3;
wire selB_pri2_1, selB_pri2_2, selB_pri2_3;
wire selC_pri2_1, selC_pri2_2, selC_pri2_3;
wire selD_pri2_1, selD_pri2_2, selD_pri2_3;
wire selA_pri3_1, selA_pri3_2, selA_pri3_3;
wire selB_pri3_1, selB_pri3_2, selB_pri3_3;
wire selC_pri3_1, selC_pri3_2, selC_pri3_3;
wire selD_pri3_1, selD_pri3_2, selD_pri3_3;
wire reqA, reqB, reqC, reqD;
wire selA, selB, selC, selD;


assign reqA = req[0];
assign reqB = req[1];
assign reqC = req[2];
assign reqD = req[3];


assign A_ge_B = (AgeA >= AgeB) ?   1'b1 :   1'b0;
assign A_lt_B =  ~A_ge_B;

assign A_ge_C = (AgeA >= AgeC) ?   1'b1 :   1'b0;
assign A_lt_C =  ~A_ge_C;

assign A_ge_D = (AgeA >= AgeD) ?   1'b1 :   1'b0;
assign A_lt_D =  ~A_ge_D;

assign B_ge_A = (AgeB >= AgeA) ?   1'b1 :   1'b0;
assign B_lt_A =  ~B_ge_A;

assign B_ge_C = (AgeB >= AgeC) ?   1'b1 :   1'b0;
assign B_lt_C =  ~B_ge_C;

assign B_ge_D = (AgeB >= AgeD) ?   1'b1 :   1'b0;
assign B_lt_D =  ~B_ge_D;

assign C_ge_A = (AgeC >= AgeA) ?   1'b1 :   1'b0;
assign C_lt_A =  ~C_ge_A;

assign C_ge_B = (AgeC >= AgeB) ?   1'b1 :   1'b0;
assign C_lt_B =  ~C_ge_B;

assign C_ge_D = (AgeC >= AgeD) ?   1'b1 :   1'b0;
assign C_lt_D =  ~C_ge_D;

assign D_ge_A = (AgeD >= AgeA) ?   1'b1 :   1'b0;
assign D_lt_A =  ~D_ge_A;

assign D_ge_B = (AgeD >= AgeB) ?   1'b1 :   1'b0;
assign D_lt_B =  ~D_ge_B;

assign D_ge_C = (AgeD >= AgeC) ?   1'b1 :   1'b0;
assign D_lt_C =  ~D_ge_C;
```

```
//
// Default Selections
//
assign selA_pri0 = (lastGrant == 'IDLE) &  reqA;
assign selB_pri0 = (lastGrant == 'IDLE) & ~reqA &  reqB;
assign selC_pri0 = (lastGrant == 'IDLE) & ~reqA & ~reqB &  reqC;
assign selD_pri0 = (lastGrant == 'IDLE) & ~reqA & ~reqB & ~reqC & reqD;


//
// Ownership Selections
//
assign selA_pri1 = (lastGrant == 'GrantA) &  reqA;
assign selB_pri1 = (lastGrant == 'GrantB) &  reqB;
assign selC_pri1 = (lastGrant == 'GrantC) &  reqC;
assign selD_pri1 = (lastGrant == 'GrantD) &  reqD;


//
// Round Robin Selections
//
assign selA_pri2_1 = (lastGrant == 'GrantB) & ~reqB & (~rot || ~reqC & ~reqD);
assign selA_pri2_2 = (lastGrant == 'GrantC) & ~reqC & (~rot & ~reqB || rot & ~reqD);
assign selA_pri2_3 = (lastGrant == 'GrantD) & ~reqD & ( rot || ~reqC & ~reqB);
assign selA_pri2 = (reqA & ~aged) & (selA_pri2_1 | selA_pri2_2 | selA_pri2_3);

assign selB_pri2_1 = (lastGrant == 'GrantC) & ~reqC & (~rot || ~reqD & ~reqA);
assign selB_pri2_2 = (lastGrant == 'GrantD) & ~reqD & (~rot & ~reqC || rot & ~reqA);
// Fix assign selB_pri2_3 = (lastGrant == 'GrantA) & ~reqA & (rot || ~reqD & ~reqC);
assign selB_pri2_3 = (lastGrant == 'GrantA) & ~reqA & rot;
assign selB_pri2 = (reqB & ~aged) & (selB_pri2_1 | selB_pri2_2 | selB_pri2_3);

assign selC_pri2_1 = (lastGrant == 'GrantD) & ~reqD & (~rot || ~reqA & ~reqB);
assign selC_pri2_2 = (lastGrant == 'GrantA) & ~reqA & (~rot & ~reqD || rot & ~reqB);
assign selC_pri2_3 = (lastGrant == 'GrantB) & ~reqB & ( rot || ~reqA & ~reqD);
assign selC_pri2 = (reqC & ~aged) & (selC_pri2_1 | selC_pri2_2 | selC_pri2_3);

assign selD_pri2_1 = (lastGrant == 'GrantA) & ~reqA & (~rot || ~reqB & ~reqC);
assign selD_pri2_2 = (lastGrant == 'GrantB) & ~reqB & (~rot & ~reqA || rot & ~reqC);
assign selD_pri2_3 = (lastGrant == 'GrantC) & ~reqC & ( rot || ~reqB & ~reqA);
assign selD_pri2 = (reqD & ~aged) & (selD_pri2_1 | selD_pri2_2 | selD_pri2_3);


//
// Age Based Selections
//
assign selA_pri3_1 = (lastGrant == 'GrantD) & ~reqD & (( rot & A_ge_B & A_ge_C) ||
                                    (~rot & C_lt_A & B_lt_A));
assign selA_pri3_2 = (lastGrant == 'GrantC) & ~reqC & (( rot & D_lt_A & A_ge_B) ||
                                    (~rot & B_lt_A & A_ge_D));
assign selA_pri3_3 = (lastGrant == 'GrantB) & ~reqB & (( rot & C_lt_A & D_lt_A) ||
                                    (~rot & A_ge_D & A_ge_C));
assign selA_pri3 = (reqA &  aged) & (selA_pri3_1 | selA_pri3_2 | selA_pri3_3);


assign selB_pri3_1 = (lastGrant == 'GrantA) & ~reqA & (( rot & B_ge_C & B_ge_D) ||
                                    (~rot & D_lt_B & C_lt_B));
assign selB_pri3_2 = (lastGrant == 'GrantD) & ~reqD & (( rot & A_lt_B & B_ge_C) ||
```

```
                                    (~rot & C_lt_B & B_ge_A));
assign selB_pri3_3 = (lastGrant == 'GrantC) & ~reqC & (( rot & D_lt_B & A_lt_B) ||
                                    (~rot & B_ge_A & B_ge_D));

assign selB_pri3 = (reqB &  aged) & (selB_pri3_1 | selB_pri3_2 | selB_pri3_3);


// Fix  assign selC_pri3_1 = (lastGrant == 'GrantB) & ~reqB & (( rot & C_ge_D & C_ge_A) ||
assign selC_pri3_1 = (lastGrant == 'GrantB) & ~reqB & (( rot & C_ge_D & C_ge_A) ||
                                    (~rot & A_lt_C & D_lt_C));
assign selC_pri3_2 = (lastGrant == 'GrantA) & ~reqA & (( rot & B_lt_C & C_ge_D) ||
                                    (~rot & D_lt_C & C_ge_B));
assign selC_pri3_3 = (lastGrant == 'GrantD) & ~reqD & (( rot & A_lt_C & B_lt_C) ||
                                    (~rot & C_ge_B & C_ge_A));
assign selC_pri3 = (reqC &  aged) & (selC_pri3_1 | selC_pri3_2 | selC_pri3_3);


assign selD_pri3_1 = (lastGrant == 'GrantC) & ~reqC & (( rot & D_ge_A & D_ge_B) ||
                                    (~rot & B_lt_D & A_lt_D));
assign selD_pri3_2 = (lastGrant == 'GrantB) & ~reqB & (( rot & C_lt_D & D_ge_A) ||
                                    (~rot & A_lt_D & D_ge_C));
assign selD_pri3_3 = (lastGrant == 'GrantA) & ~reqA & (( rot & B_lt_D & C_lt_D) ||
                                    (~rot & D_ge_C & D_ge_B));
assign selD_pri3 = (reqD &  aged) & (selD_pri3_1 | selD_pri3_2 | selD_pri3_3);

assign selA = selA_pri0 | selA_pri1 | selA_pri2 | selA_pri3;
assign selB = selB_pri0 | selB_pri1 | selB_pri2 | selB_pri3;
assign selC = selC_pri0 | selC_pri1 | selC_pri2 | selC_pri3;
assign selD = selD_pri0 | selD_pri1 | selD_pri2 | selD_pri3;

endmodule // dpath
```

age.v

```
//   ********************************************************************************
//
//File:age.v
// Description:Determines the Age of a Request.
//
//   ********************************************************************************
//

module age (ck, reset, req, ack, AgeA, AgeB, AgeC, AgeD);
inputck, reset;
input[3:0] req, ack;
output[2:0] AgeA, AgeB, AgeC, AgeD;

wireReqA, ReqB, ReqC, ReqD;
wireAckA, AckB, AckC, AckD;
wireAwon, Bwon, Cwon, Dwon;
reg[2:0] AgeA, AgeB, AgeC, AgeD;

assign Awon = req[0] && ack[0];
assign Bwon = req[1] && ack[1];
assign Cwon = req[2] && ack[2];
assign Dwon = req[3] && ack[3];

always @(posedge ck)
```

```
begin
    if (reset)
          begin
AgeA <= 3'd0;
AgeB <= 3'd0;
AgeC <= 3'd0;
AgeD <= 3'd0;
          end
    else
       if (Awon)
          begin
 AgeA <= 3'd0;
             if (req[1])
                begin
             if (AgeB < 6)
              AgeB <= AgeB + 3'd1;
             else
              AgeB <= AgeB;
                end
             if (req[2])
                begin
             if (AgeC < 6)
              AgeC <= AgeC + 3'd1;
             else
              AgeC <= AgeC;
                end
             if (req[3])
                begin
             if (AgeD < 6)
              AgeD <= AgeD + 3'd1;
             else
              AgeD <= AgeD;
                end
          end
       if (Bwon)
          begin
 AgeB <= 3'd0;
             if (req[0])
                begin
             if (AgeA < 6)
              AgeA <= AgeA + 3'd1;
             else
              AgeA <= AgeA;
                end
             if (req[2])
                begin
             if (AgeC < 6)
              AgeC <= AgeC + 3'd1;
             else
              AgeC <= AgeC;
                end
             if (req[3])
                begin
             if (AgeD < 6)
              AgeD <= AgeD + 3'd1;
             else
              AgeD <= AgeD;
                end
          end
```

```
        if (Cwon)
           begin
 AgeC <= 3'd0;
                if (req[0])
                   begin
                if (AgeA < 6)
                 AgeA <= AgeA + 3'd1;
                else
                 AgeA <= AgeA;
                   end
                if (req[1])
                   begin
                if (AgeB < 6)
                 AgeB <= AgeB + 3'd1;
                else
                 AgeB <= AgeB;
                   end
                if (req[3])
                   begin
                if (AgeD < 6)
                 AgeD <= AgeD + 3'd1;
                else
                 AgeD <= AgeD;
                   end
             end
        if (Dwon)
           begin
 AgeD <= 3'd0;
                if (req[0])
                   begin
                if (AgeA < 6)
                 AgeA <= AgeA + 3'd1;
                else
                 AgeA <= AgeA;
                   end
                if (req[1])
                   begin
                if (AgeB < 6)
                 AgeB <= AgeB + 3'd1;
                else
                 AgeB <= AgeB;
                   end
                if (req[2])
                   begin
                if (AgeC < 6)
                 AgeC <= AgeC + 3'd1;
                else
                 AgeC <= AgeC;
                   end
             end
end
endmodule // age
```

VERIFICATION RESULTS:

Query-1:

Query: NoAck_Unless_Req

PROPERTIES:

Property: NoAcks
  Type: Always

  Always: (arb.ack == 0)
  Unless After: (arb.req >= 1) && (arb.ck == rising)

  Options: Fulfill at discharge

CONSTRAINTS:

Clock Constraint: ck
  Signal: arb.ck
  Extract: No
  Default: Yes

  Start: Low
  1st Duration: 1
  2nd Duration: 1

Reset Constraint: reset
  Signal: arb.reset
  Default: Yes

  Start: High

  Transition Duration Value
      Start   2   1
            forever   0

RUN OPTIONS:

  Algorithm: Symbolic (BDD)

REDUCTION OPTIONS:

  Reduction Technique: 1-Step
  Reduction Seed: Empty

RESULT:

Verification Query NoAck_Unless_Req VERIFIED! Wed May 10 01:15:50 2000 on server: richards

Query Data:
1.32e3 combinational variables
1.02e3 Possible input combinations per state
20 State variables: 2.1e+06 states

Verification Data:
5 states reached.
State variable coverage: 20 variables, 57.50% average coverage
Search Depth: 5
Real time: 0 minutes 23 seconds
Memory Usage: 5.99654 megabytes

EXPRESSION MACROS:

@AckA: arb.ack[0] == 1
@AckB: arb.ack[1] == 1
@AckC: arb.ack[2] == 1
@AckD: arb.ack[3] == 1
@All_Reqs: arb.req == 15
@CkRising: arb.ck == rising
@Multi_Acks: (((arb.ack[3] + arb.ack[2]) + arb.ack[1]) + arb.ack[0]) > 1
@ReqA: arb.req[0] == 1
@ReqB: arb.req[1] == 1
@ReqC: arb.req[2] == 1
@ReqD: arb.req[3] == 1
@ResetDone: arb.reset == finished

Query-2:

Query: Only_One_Ack

There are 4 request inputs and 4 acknowledge outputs in this design.

We want to verify that the arbiter never acknowledges more than one request at a time.

Ensure P1{NEVER{Ack1 + Ack2 + ... + Ackn >1}}

THIS IS A NEVER PROPERTY.

PROPERTIES:

Property: Never_Multiple_Acks
  Type: Never

  Never: @Multi_Acks

  Options:(None)

CONSTRAINTS:

Clock Constraint: ck
  Signal: arb.ck
  Extract: No
  Default: Yes

  Start: Low
  1st Duration: 1
  2nd Duration: 1

Reset Constraint: reset
  Signal: arb.reset
  Default: Yes

  Start: High

  Transition Duration Value
      Start   2   1
            forever   0

RUN OPTIONS:

  Algorithm: Symbolic (BDD)

REDUCTION OPTIONS:

  Reduction Technique: 1-Step
  Reduction Seed: Empty

RESULT:

Verification Query Only_One_Ack VERIFIED! Wed May 10 01:16:29 2000 on server: richards

Query Data:
1.32e3 combinational variables
1.02e3 Possible input combinations per state
20 State variables: 2.1e+06 states

Verification Data:
1.11e+04 states reached.

State variable coverage: 20 variables, 97.50% average coverage
Search Depth: 20
Real time: 0 minutes 17 seconds
Memory Usage: 5.99654 megabytes

Query-3:

ery: AckA_Width

Checking if the AckA is minimum of 2 clocks wide.

We can later check for AckB, AckC and AckD. Which can be easily done in command line version.

PROPERTIES:

Property: Min_Width_AckA_2clks
Type: Always

After: arb.ack[0] == rising
Always: arb.ack[0] == 1

Options: Fulfill Delay: 0 Duration: 2 counts of
@CkRising

CONSTRAINTS:

Constant Constraint: Dont_bypassA
Signal: arb.byp_req[0]
Default: No
Value: 0

Group Constraint: ReqA
Constraints: ReqA_cant_occur
ReqA_persists

Default: Yes

Clock Constraint: ck
Signal: arb.ck
Extract: No
Default: Yes

Start: Low
1st Duration: 1
2nd Duration: 1

Reset Constraint: reset

Signal: arb.reset
Default: Yes

Start: High

Transition Duration Value
Start 2 1
forever 0

RUN OPTIONS:

Algorithm: Symbolic (BDD)

REDUCTION OPTIONS:

Reduction Technique: 1-Step
Reduction Seed: Empty

RESULT:

Verification Query AckA_Width VERIFIED! Wed May 10 01:17:03 2000 on server: richards

Query Data:
1.32e3 combinational variables
512 Possible input combinations per state
25 State variables: 1e+08 states

Verification Data:
1.54e+04 states reached.
State variable coverage: 25 variables, 96.00% average coverage
Search Depth: 42
Real time: 0 minutes 24 seconds
Memory Usage: 6.0375 megabytes

Query-4:

Query: Acka

This query includes a liveness property and fairness constraint.

It says that eventually there will be an acknowledgment for the request from A.

PROPERTIES:

Property: Eventually_AckA
  Type: Eventually

  After: @ResetDone && @ReqA && @CkRising
  Eventually: @AckA

  Options:(None)

CONSTRAINTS:

Constant Constraint: Dont_bypassA
  Signal: arb.byp_req[0]
  Default: No
  Value: 0

Group Constraint: ReqA
    Constraints: ReqA_cant_occur
  ReqA_persists

  Default: Yes

Group Constraint: ReqB
    Constraints: ReqB_cant_occur
  ReqB_persists

  Default: Yes

Group Constraint: ReqC
    Constraints: ReqC_cant_occur
  ReqC_persists

  Default: Yes

Group Constraint: ReqD
    Constraints: ReqD_cant_occur
  ReqD_persists

  Default: Yes

Constraint Stable_Rotation
  Type: Always

  After: @ResetDone && @CkRising
  Assume Always: arb.rot == stable

  Options: (None)

Constant Constraint: am_zero
  Signal: arb.am
  Default: No
  Value: 0

To make the arbitration mode non age based

```
--------------------------------------------------------------------------------
Clock Constraint: ck
  Signal: arb.ck
  Extract: No
  Default: Yes

  Start: Low
  1st Duration: 1
  2nd Duration: 1
--------------------------------------------------------------------------------
Reset Constraint: reset
  Signal: arb.reset
  Default: Yes

  Start: High

  Transition Duration Value
      Start       2     1
            forever     0

--------------------------------------------------------------------------------


RUN OPTIONS:
--------------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
--------------------------------------------------------------------------------

REDUCTION OPTIONS:
--------------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
--------------------------------------------------------------------------------

RESULT:
--------------------------------------------------------------------------------
Verification Query Acka VERIFIED! Wed May 10 01:17:44 2000 on server: richards

Query Data:
557 combinational variables
256 Possible input combinations per state
16 State variables: 6.55e4 states
 **** Same as prior verification Wed May 10 00:10:16 2000

Verification Data:
Reachable space: 495 states
495 states reached.
State variable coverage: 19 variables, 100.00% average coverage
Search Depth: 10
Real time: 0 minutes 11 seconds
Memory Usage: 2.32653 megabytes.
```

Query-5:

uery: Seq_ClockwiseAged

This is a Liveness property.

To verify that the sequence for Clockwise/Aged arbitration is

AckA -> AckB -> AckC -> AckD

Also... note the use of state variable called "Seq".

PROPERTIES:

Property: ClockwiseAged
Type: Eventually

After: @All_Reqs && (Seq == 0) && @CkRising && (arb.ack == 0)
Eventually: Seq == 4

Options:(None)

CONSTRAINTS:

Group Constraint: ReqA
Constraints: ReqA_cant_occur
ReqA_persists

Default: Yes

Group Constraint: ReqB
Constraints: ReqB_cant_occur
ReqB_persists

Default: Yes

Group Constraint: ReqC
Constraints: ReqC_cant_occur
ReqC_persists

Default: Yes

Group Constraint: ReqD
Constraints: ReqD_cant_occur
ReqD_persists

Default: Yes

Constant Constraint: am_high

Signal: arb.am
Default: No
Value: 1

Constant Constraint: byp_req_low
Signal: arb.byp_req
Default: No
Value: 0

Clock Constraint: ck
Signal: arb.ck
Extract: No
Default: Yes

Start: Low
1st Duration: 1
2nd Duration: 1

Reset Constraint: reset
Signal: arb.reset
Default: Yes

Start: High

Transition Duration Value
Start 2 1
forever 0

Constant Constraint: rot_high
Signal: arb.rot
Default: No
Value: 1

STATE VARIABLES:

Seq: Range 0 to 4
Initial: 0

```
if ((Seq == 0) && @AckA && @CkRising)
   Seq = 1;
else if ((Seq == 1) && @AckB && @CkRising)
   Seq = 2;
else if ((Seq == 2) && @AckC && @CkRising)
   Seq = 3;
else if ((Seq == 3) && @AckD && @CkRising)
   Seq = 4;
else if ((Seq == 4) && @CkRising)
   Seq = 0;
else
   Seq = Seq;
```

RUN OPTIONS:

Algorithm: Symbolic (BDD)

REDUCTION OPTIONS:

Reduction Technique: 1-Step
Reduction Seed: Empty

RESULT:

Verification Query Seq_ClockwiseAged VERIFIED! Wed May 10 01:18:12 2000 on server: richards

Query Data:
925 combinational variables
16 Possible input combinations per state
28 State variables: 6.71e8 states
**** Same as prior verification Wed May 10 00:48:10 2000

Verification Data:
Reachable space: 2.86e+03 states
2.86e+03 states reached.
State variable coverage: 30 variables, 93.33% average coverage
Search Depth: 42
Real time: 0 minutes 12 seconds
Memory Usage: 2.32653 megabytes.

EXPRESSION MACROS:

@AckA: arb.ack[0] == 1
@AckB: arb.ack[1] == 1
@AckC: arb.ack[2] == 1
@AckD: arb.ack[3] == 1
@All_Reqs: arb.req == 15
@CkRising: arb.ck == rising
@Multi_Acks: (((arb.ack[3] + arb.ack[2]) + arb.ack[1]) + arb.ack[0]) > 1
@ReqA: arb.req[0] == 1
@ReqB: arb.req[1] == 1
@ReqC: arb.req[2] == 1
@ReqD: arb.req[3] == 1
@ResetDone: arb.reset == finished